University of Kent

PhD Thesis

Entropy-Based Resource Management in Complex Cloud Environment

Author:
Huankai Chen

Supervisor:
Prof F Z Wang

Supervision Team Members:
Prof Sally Fincher , Dr E A Boiten, Dr M Migliavacca

A thesis submitted by Huankai Chen to the University of Kent for the degree of Doctor of Philosophy

in

Computer Science

October 2016

UNIVERSITY OF KENT

Abstract

Faculty of Sciences
School of Computing

Doctor of Philosophy

Entropy-Based Resource Management in Complex Cloud Environment

by Huankai CHEN

Resource Management is an NP-complete problem, the complexity of which increases substantially in the Cloud environment. The complexity of cloud resource management can originate from many factors: the scale of the resources; the heterogeneity of the resource types and the interdependencies of these; as well as the variability, dynamicity and unpredictability of resource run-time performance.

Complexity has many negative effects in relation to satisfying the Quality of Service (QoS) requirements of cloud applications, such as cost, performance, availability and reliability. If an application cannot guarantee its QoS, it will be hard to populate. However, the vast majority of research efforts into cloud resource management implicitly assume the Cloud to be a simplifying technology and that the cloud resource's performance is determined and predictable. These incorrect assumptions may significantly affect the QoS of any cloud application developed under it, causing its resource management strategy to be less than robust.

In spite of there being extensive research into complexity issues in many diverse fields ranging from computational biology to decision making in economies, the study of complexity in cloud resource management systems is limited. In this thesis, I address the complexity problems of *Cloud Resource Management Systems* by introducing the use of **Entropy Theory** in relation to them. The main contributions of this thesis are as follows:

1. A cloud simulation tool-kit, **ComplexCloudSim**, is implemented in order to help tackle the research question: what is the role of complexity in QoS-aware cloud resource management?

2. The uncovering of **Chaotic Behaviour** in *Cloud Resource Management Systems* by using the **Damage Spreading Analysis** method.

3. The comprehensive definition of complexity in the *Cloud Resource Management Systems*; such can be primarily classified into two categories: **Global System Complexity** and **Local Resource Complexity**.

4. An **Entropy Theory** based resource management model is proposed for the purposes of identifying, measuring, analysing and controlling (i.e., reducing and avoiding) complexity.

5. An **Cellular Automata Entropy** based methodology is proposed as a solution to the Cloud resource allocation problem; this methodology is capable of managing **Global System Complexity**.

6. Once the root cause of the complexity has been identified using the Local Activity Principle, a **Resource Entropy Based Local Activity Ranking** system can be proposed which solves the job scheduling problem by managing **Local Resource Complexity**. Finally, on this latter basis, I implement a system which I have termed an **Entropy Scheduler** within a popular real-world cloud analysis engine, Apache Spark. Experiments demonstrate that the new **Entropy Scheduler** significantly reduces the average query response time by 15% - 20% and standard deviation by 30% - 45% compare with the native Fair Scheduler for running CPU intensive applications in Apache Spark, when the Spark server is not overloaded.

Acknowledgements

The study for PhD is intense and full of challenges. I could not have achieved anything without help from many people.

First and foremost, I offer my sincerest gratitude to my supervisor, Professor Frank Wang, who supported me throughout my thesis with his patience and guidance whilst allowing me the room to work in my own way. For me, he was not only a respectable scientist who led me on the way to do research, but also an attentive tutor who trained me to be a good professor in my future career. I really appreciate everything he has done in the past years.

I am also grateful to my supervision team members, Professor Sally Fincher, Dr E A Boiten and Dr M Migliavacca, for their valuable comments on my thesis.

Special thanks should be given to Professor Leon O. Chua who joined School of Computing, University of Kent, as an EC Marie Curie Fellow on 1 August 2013. His professional abilities and knowledge are always admired. When I encountered the obstacles in research, the discussion with him often inspired me, theoretically or practically. I greatly thank him for all the kindness and support.

Discussions with fellow students and researchers in the Future Computing Group, especially Xiao Yang, Wanlong Chen, Mian-Guan Lim, Gbola Akanmu, Saad Alshahrani and Chen Hu, were stimulating and entertaining. I'm thankful for their friendship and help.

And I wish to express my gratitude to Dr Na Helian for her support and for being there whenever I needed her help and feedback.

Last but not least, I would like to dedicate this thesis to my parents, families and the people I love and who love me. And most importantly, I would like to thank my beloved wife Jia Jia. I would not have been able to go through this without her support. Together, we have made a great journey so far. I am sure that it will be even greater from now on.

Contents

List of Publications

1. *Huankai Chen, Frank Z. Wang, Na Helian and Gbola Akanmu.* "***User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing.***" *Parallel Computing Technologies (PARCOMPTECH), 2013 National Conference on. IEEE, 2013.*

2. *Huankai Chen, Frank Z. Wang, and Na Helian.* "***A Cost-Efficient and Reliable Resource Allocation Model Based on Cellular Automata Entropy for Cloud Project Scheduling.***" *International Journal of Advanced Computer Science and Applications. 05/2013; 4(4):7-14.*

3. *Huankai Chen, and Frank Z. Wang.* "***Spark on entropy: A reliable & efficient scheduler for low-latency parallel jobs in heterogeneous cloud.***" *Local Computer Networks Conference Workshops (LCN Workshops), 2015 IEEE 40th. IEEE, 2015.*

4. *Huankai Chen, Frank Z. Wang, Matteo Migliavacca, Leon O. Chua, and Na Helian.* "***Complexity Reduction: Local Activity Ranking By Resource Entropy For QoS-aware Cloud Scheduling.***" *13th IEEE International Conference on Services Computing. IEEE, 2016.*

5. *Huankai Chen, and Frank Z. Wang.* "***ComplexCloudSim: Towards Understanding Complexity in QoS-aware Cloud Scheduling.***" *International Journal of Advanced Computer Science and Applications. 03/2017; 8(3):9-16.*

6. *Huankai Chen, Frank Z. Wang and Na Helian.* "***Entropy4Cloud: Using Entropy-Based Complexity to Optimize Cloud Service Resource Management.***" *IEEE Transactions on Emerging Topics in Computational Intelligence. 02/2018; 2(1):13-24.*

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing is everywhere. When we look at any IT related magazine, website, or TV programme, the word "Cloud" will almost certainly catch our eye. All of today's most popular social networking, email, document-sharing and on-line gaming sites are hosted on a cloud. Even the U.K. government intends to transform the public sector ICT estate into one that is agile, cost effective and environmentally sustainable by exploiting innovations in Cloud computing [GOV.UK, 2011]. Cloud computing can make a software system more attractive as a service, and shapes the way in which IT hardware is purchased. It is possible to predict that it will spark a revolution in the way organizations provide and consume information and computing.

Cloud computing has reached into our daily lives and has led to a broad range of innovations. Built on a number of older IT technologies, cloud computing is actually a revolutionary approach that completely changes how computing services are produced, priced and delivered. It allows users to access services that reside at a distant data centre, rather than on local computers or on other Internet-connected devices. Cloud services are charged according to the amount consumed by users. The idea of computing services as easily accessible utilities has been a long-held dream in the computer industry, but the idea is still not mature, and will not become so until the advent of low-cost and reliable data centres.

Data centres behaving as "cloud providers" are computing infrastructures which provide many kinds of flexible and effective services to customers. A wide range of IT companies, including Amazon, Cisco, Yahoo, Salesforce, Facebook, Microsoft and Google have their own data centres and provide pay-as-you-go cloud services. Two different but related types of cloud service should be distinguished from each other first: the on-demand computing instance, and the on-demand computing capacity. Equipped with similar

machines, data centres can scale out by providing additional computing instances, or they can support data- or compute-intensive applications via scaling capacity.

Windows Azure, Amazon Web Services, the Google App Engine and Force.com are examples of the first category; they provide computing instances according to needs. The data centres instantly create virtualized instances and provide a response. The virtualized instance might consist of processors running at different speeds that span different storage systems at different locations. Therefore, virtualization is an essential characteristic of Cloud computing; through virtualization applications can be executed independently without regard to any particular configuration.

Google and Yahoo belong to the second category. At these data centres, the need to process large amounts of raw data is primarily met by distributed and parallel computing, and the data can be moved from place to place and assigned changing attributes based on its life-cycle, requirements and usefulness. One core technology is MapReduce, a style of parallel programming model supported by capacity-on-demand clouds. It can deal with massive data in parallel on a cloud.

The above two types of cloud services classify cloud computing into two distinct deployment models: public and private. A public cloud is designed to provide cloud services to a variety of third-party clients who use the same cloud resources. Public cloud services such as Googles App Engine are open to anyone at any time and anywhere. On the contrary, a private cloud is devoted to a single organization's internal use. Google, for example, uses GFS, MapReduce, and BigTable within its private cloud services, so the services provided by these systems are only available inside the enterprise. It's important to note that Google uses its private cloud to provide its public cloud services, such as those related to productive applications, media delivery, and social interaction.

1.1 Motivations and Challenges

Cloud computing is still in its infancy, but it has presented new opportunities to users and developers who can benefit from economies of scales, commoditization of assets and conformance to programming standards. Its attributes, such as scalability, elasticity, low barrier to entry and a utility type of delivery make this new paradigm readily marketable.

At the same time, the cloud market poses a number of challenges. Resource management is one of theses and is a topic very worthy of investigation; it is a key issue in the determination of whether the new computing paradigm will be adopted more widely and will meet with even greater business success.

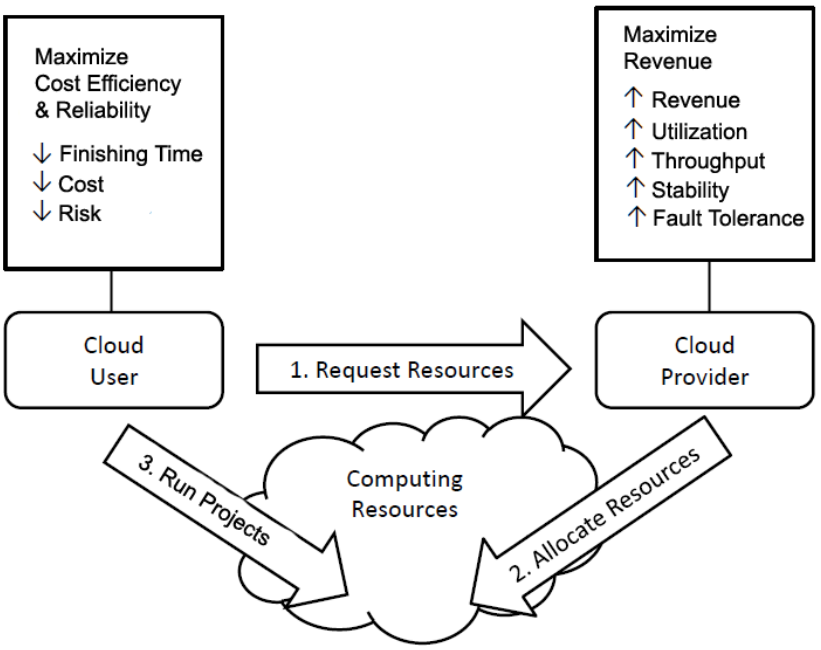

FIGURE 1.1: Cloud Usage Scenario.

In most cases, the interaction between cloud users and cloud providers occurs as shown in 1.1. First, a user sends a request for resources to a provider. When the provider receives the request, it looks for resources to satisfy the request and assigns the resources to the requesting user, typically in the form of virtual machines (VMs). Then the user uses the assigned resources in order to run projects and pays for the resources that are used. When the user no longer needs the resources, they are returned to the provider. One interesting aspect of the resource management problem which concerns the cloud market is that the two players, cloud users and cloud providers, are often different parties with their own divergent interests.

- From the cloud user's resource management perspective, the goal is to deliver user's projects within deadline and cost budget. In other words, the cloud user seeks to maximize their cost performance with reliable resource management solutions. This involves renting the proper quantity of cloud resources to suit the project requirements and utilize the resources effectively. Also, unpredictable resources production/performance has to be taken into account when projects are scheduled and resources are requested.

- From the cloud provider's resource management perspective, the focus is more on system performance and the revenue of the . In this case, improvement from resource management mainly relates to technical issues. For example, it is important to optimize the scheduling schemes in order to reduce project completion

time and to improve resource utilization - when many users' projects are running in parallel. To generate as much revenue as possible with minimum investment, a cloud provider might try to put a strain on their computing resources by, for example, hosting as many VMs as possible on each machine. However, placing too many VMs on a single machine will cause VMs to interfere with each other, resource-wise, and may result in degraded and/or unpredictable production performance, and this, in turn, may frustrate the users. Thus, the provider could evict existing VMs or reject resource requests in order to maintain service quality, but this might make the cloud environment even more unpredictable.

However, these two parties are generally not able to share information with each other efficiently, which makes optimal resource allocation and scheduling more difficult. For example, how many of what kind of machines the providers have and how these are connected is information which is hidden from the users. Similarly, providers cannot allocate resources in a manner most suitable to users' projects, since there is no information available to them about the workload pattern these will exhibit. It is also difficult for providers to multiplex their resources effectively when they have to assign resources to projects with heterogeneous (and unknown) types of workload pattern.

In addition, data centres are becoming increasingly heterogeneous and may consist of various generations of equipment, as the technology advances. For example, processors with more cores and greater cache memory are continuously being introduced onto the market.

Moreover, due to its increasing complexity, the cloud environment is highly unpredictable. Sometimes, cloud providers even voluntarily offer more unpredictable resource containers at a lower cost. For example, Amazon's EC2 offers spot instances for which users make a bid that is much lower than the price of regular instances. If the load to EC2 surges and the price of spot instances spikes higher than the bid, then these instance become liable to be evicted at any time. Thus, there are no guarantees for users running projects on spot instances.

Hence, we must take these properties of the cloud environment into account to make cloud services efficient. By efficient, from the cloud providers' point of view, we mean that appropriate resources are allocated at the right time to the right project, so that the users' projects can utilize the available resources effectively. From the cloud user's point of view, we want to minimize the amount of resources needed for a project to maintain a desirable level of service quality that meets the project's QoS requirements, such as its deadline, cost budget and reliability requirements. However, we argue that

many current cloud resource management solutions are inefficient due to the following reasons:

- The lack of information sharing
- The assumption of homogeneous cloud environments
- Highly dynamic cloud resource performance
- The complexity of cloud resource management increased due to QoS constraints

1.2 Problem Statement and Contribution

In Cloud computing, a cloud service's resource management system usually contains two levels of processes: resource allocation and job scheduling.

- **Resource Allocation** is the process of finding a suitable quantity of resources for the Cloud application while meeting the application's QoS requirements. At this level, it is very difficult to quantify the performance of an allocation policy on cloud infrastructures for applications exhibiting varying workloads and resource usages. The simulation based approaches provide significant benefits, as they allow researchers to test their proposed algorithms and protocols in a repeatable and controlled environment in order to find solutions to any performance bottlenecks associated with these, before deploying in the real Cloud. However, most of the current cloud simulators lack an accurate perspective on the actual nature of the Cloud environment, **complexity**, and this makes them incapable of modelling real-world complex cloud scenarios. Inaccurate simulations may result in the over-provisioning or under-provisioning of resources for Cloud applications.
- **Job Scheduling** focuses on the mapping of application jobs onto the available resources. In the cloud environment, the computational complexity of finding an optimum resource mapping is exponential since cloud resources' performances can be highly dynamic and uncertain during run-time. Thus, most of the current research solutions ignore the complexity factors involved with cloud resources since their performances are nearly entirely unpredictable. Such studies assume cloud resources performance to be unchanging during the run-time of the application and only focus on the static information available (e.g., the number of CPU cores) when developing new scheduling strategies. Such incorrect assumptions make their scheduling strategies less robust when running on the complex, real world Cloud.

This thesis studies the above resource management problems related to complex Cloud environments and tries to tackle these problems by introducing the use of the concept of Entropy Theory into this area. The major contributions of this thesis are as follows.

- A survey of current trends and research opportunities in Cloud computing.
- A simulator for modelling complexity in Cloud environments, to facilitate the study of the impacts of complexity on cloud resource management systems.
- The introduction of Entropy Theory as concept to assist the understanding of resource management in the complex cloud environment
- The proposal of a cloud resource allocation methodology based on Cellular Automata Entropy
- The proposal of a cloud job scheduling strategy guided by Local Activity Ranking as measured by Resource Entropy

1.3 Thesis Organization

The rest of this thesis is organized as follows.

- **Chapter 2** provides a comprehensive survey of resource management research in respect of the Cloud. The state-of-the-art efforts on cloud resource management systems are investigated from both industry and academic standpoints. Here I also identify previous work that is related to my work and highlight the resource management issues that deserve further substantial research and development.
- **Chapter 3** evaluates the impact of complexity in cloud resource management systems. I present **ComplexCloudSim**, which extends the popular simulation tool-kit CloudSim with the ability to model the complexity factors involved in the Cloud including heterogeneity of resources, dynamic changes of resource runtime performance and uncertainty of task execution time. Furthermore, damage spreading analysis is applied to the area of cloud resource management systems. The simulations show that cloud systems reveal sensitivity to initial conditions in some parameter regions. Such findings of "**Chaotic Behaviour**" explain why most cloud resource management systems work less robustly in the real world production environment; this is a problem which cannot be ignored.
- **Chapter 4** firstly tries to define the complexity involved with the cloud resource management systems. This is classified into two general types: *Global System*

Complexity and *Local Resource Complexity*. **Entropy Theory** is then introduced as a concept which will assist in the managing of the complexity of cloud resource management system and their issues - this covers identifying, measuring, analysing and controlling.

- **Chapter 5** proposes solutions to the resource allocation problem based on managing *Global System Complexity*. First, I present a short tutorial on *Cellular Automata*, covering the different behaviour classes of Cellular Automata and also Entropy as a quantitative measurement which can be used to identify such classes. Next, **Cellular Automata Entropy based resource allocation (CAE-CRA)**, a methodology, is proposed to better satisfy the QoS requirements of cloud projects. Finally, the proposed methodology is implemented under the Matlab environment and verified in relation to four basic resource allocation strategies, the First Come First Served Algorithm (FCFS), the Round Robin Algorithm (RR), the Min-Min Algorithm and the Max-Min Algorithm. The experimental results show that the proposed methodology leads to the attainment of more cost-efficient and reliable cloud resource allocation solutions.

- **Chapter 6** proposes solutions to the Cloud job scheduling problem based on managing *Local Resource Complexity*. We first study the concept of the *Local Activity Principle* and also several complexity factors caused by the locally-active Cloud resource. And then I propose **Local Activity Ranking by Resource Entropy** as a methodology to control the *Chaos* in QoS-aware cloud job scheduling. Finally, this concept, **Entropy Scheduler**, is implemented in a widely-used real-world cloud analysis engine Apache Spark. Experiments demonstrate that the new *Entropy Scheduler* can gain significant improvements on the satisfaction of QoS requirement comparing with the *native Spark Fair Scheduler*.

- **Chapter 7** concludes the whole thesis.

Chapter 2

Literature Review: Cloud Resource Management System

This chapter begins with a general introduction to Cloud computing and Cloud applications which are concerned with MapReduce-like jobs. After that, I investigate related works, both from industry and research perspectives, regarding the resource allocation and job scheduling problem. Finally, the challenges and opportunities in this domain are captured.

2.1 Background

Over the past decade, Cloud computing has profoundly changed the way people use resources and services. Below is the definition of Cloud computing given by the National Institute of Science and Technology (NIST) [Liu et al., 2011a]:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

Cloud computing encompasses both: a) the provision of resources to consumers on a pay-per-use or charge-per-use basis, and b) the private infrastructures maintained and utilized by individual consumers. The former case is referred to as the *Public Cloud* and the latter as a *Private Cloud*. The scenario wherein a consumer extends the capacity of their private Cloud infrastructure by renting out spare public Cloud resources is referred

to as *Hybrid Cloud*. Another emerging category is *Community Cloud*, where the Cloud resources are contributed by many individuals/organisations and where governance is decentralised.

Cloud environments are typically classified into the following service models [Liu et al., 2011a]:

- Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources using which the consumer is able to deploy and run arbitrary software, including operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over their operating systems, storage, and deployed applications - and possibly limited control of selected networking components (e.g., host firewalls).
- Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or consumer-acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, and storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.
- Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, storage, or even individual application capabilities with the possible exception of limited user-specific application configuration settings.

Throughout this thesis, I refer to IaaS as "the Cloud environments" or "the Cloud" unless otherwise specified. SaaS and PaaS are also important service models of Cloud computing, but they are not within the focus of this thesis.

The last few years have seen a dramatic growth in the availability of, and the demand for, the Cloud. Nowadays, a wide range of IT companies, including Amazon, Microsoft, Google, IBM, HP and Rackspace, provide Infrastructure as a Service (IaaS). The IaaS provider owns and maintains the data centre while the consumers rent out the specific physical/virtual resources they need for deploying their cloud applications - usually on a "pay as you go" basis. With advanced virtualization technologies, a single physical host

can run multiple virtual machines (VMs) simultaneously. Nevertheless, this virtualization also brings about new challenges to resource management for Cloud applications due to the fact that multiple VMs can share the hardware resources (e.g. CPU, memory, I/O, network, etc.) of a physical machine. In such situations, it is difficult to accurately measure the actual performance of rented VMs. For example, in Amazon EC2, the provisioning of resources to virtual machines is based on compute units instead of fixed performance measures. Different host machines provide a different amount of computing power per provisioned compute unit, effectuating heterogeneity among VM performance.

2.1.1 Cloud Applications Consists of MapReduce Jobs

MapReduce is a programming model for processing parallelizable computing across huge amounts of data using a number of resources - collectively referred to as the Cloud. Typically, the data is broken down into smaller pieces (called blocks) and distributively stored across a distributed file system: e.g. the Hadoop Distributed File System (HDFS). MapReduce allows for the distributed processing of the map and reduction operation on the data, which provides the scalability which is needed for big data processing in Cloud applications.

A MapReduce job centres around two functions: a map function and a reduce function. Fig. 2.1 briefly describes the workflow in a MapReduce job. Each Map function loads a block of input data and generates corresponding intermediate data that is grouped by keys. Then the intermediate data is sent to the corresponding reduce function which is responsible for a range of the key space and produces the final output.

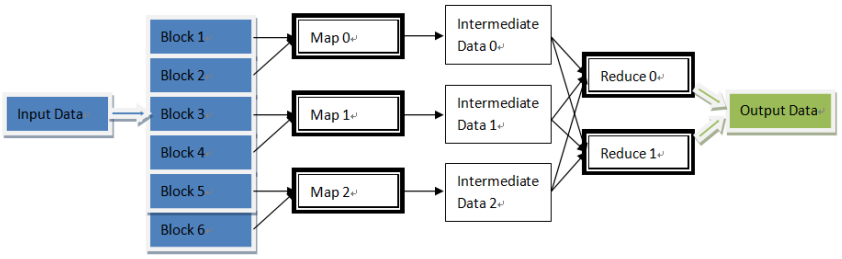

FIGURE 2.1: Logical View of MapReduce Job

2.1.2 Resource Management System for Cloud Applications

In Cloud applications consisting of MapReduce jobs, there is a single *Master* node that manages the whole Cloud. A number of *Worker* nodes subscribe to the *Master* node

in order to join the cloud. Each *Worker* may have one or more slots, depending on the number of allocated CPU cores, and each slot runs one map or reduce function at a time.

The *Master* node contains a Resource Management System (RMS), either internal (e.g. Spark Standalone) or external (e.g. Yarn, Mesos). The RMS typically contains three major functionalities to manage the cloud as follows:

- The **Resource Allocation** function is responsible for allocating resources (Physical/Virtual machines) to the cloud application subject to the familiar constraints of capacities, queues etc. and deploying the *Worker* nodes on the allocated resources. For this function, the user needs to make a decision on choosing a suitable number of resources, according to the application's requirement in a-priori.
- The **Job Scheduling** function is responsible for scheduling MapReduce jobs in the deployed *Worker*s' slots. When the scheduler receives a MapReduce job, it will first check whether there are enough qualified slots to run the job. If the scheduler finds that the number of available slots meets the job's requirement, it assigns the job's Map/Reduce tasks to the slots; such slots will become unavailable until all the tasks assigned are finished.
- **Resource & Job Monitoring** is responsible for monitoring the resource usage (cpu, memory, disk and network) and the progress of running jobs.

2.2 Related Works

In this section, I describe in greater detail the two major functions of a RMS in relation to a cloud application: Resource Allocation and Job Scheduling. We also highlight the previous research, the current status of research and future directions in these domains.

2.2.1 Resource Allocation

The objective of resource allocation is to find and allocate the appropriate resources to the cloud applications on time, so that applications can utilize these resources effectively. Moreover, the amount of resources should be the minimum for a cloud application to maintain its desired level of service quality, including performance, availability, scalability, cost budget, security, etc. Typically, allocating resources to an application is a static process which occurs prior to the running of the application. To maximize revenues and

improve user satisfaction, an effective allocation of resources to cloud applications is necessary.

The functionality of resource allocation can be divided into two main components:

1. determine both the amount of resources a cloud application needs, and the specific details of the resources required,
2. make and enforce resource allocation decisions in a way that satisfies the resource requirements of the cloud application.

For both components there is a wide spectrum of designs and implementations. The fundamental kind of resource allocation found from the existing literature can, in the main, be divided into the following categories: QoS based, resource based, bargaining based, prediction based and nature-inspired/bio-inspired.

2.2.1.1 QoS (e.g. Budget, Deadline, Reliability) Based

Isard et al. [2009] formulates resource assignment as a graph optimization problem, accounting for fairness and the placement constraints applications may have. A formulation that supports a mix of QoS scenarios with precisely defined objective function, promotes performance, fairness, and CPU utilization is proposed for static workloads with multiple types of resources by Stillwell et al. [2010]. Byun et al. [2011] propose an architecture to automatically execute large-scale workflow-based applications on dynamically and elastically provisioned cloud resources. Sharma et al. [2011] present a cost-aware resource allocation system that optimize the selection of virtual server configuration to minimizes the cost. Hwang and Kim [2012] propose a cost-effective resource provisioning methodology for deadline constrained cloud applications. A approach that operates fine-gained resource level scaling as well as VM level scaling (CPUs, Memory, I/O) is proposed to support cost-effective elasticity for cloud services by Han et al. [2012]. Mao and Humphrey [2011] present an approach to ensure all jobs are finished within deadlines at lowest financial cost, where takes the virtual machine of various sizes/costs as the basic computing units and which (soft) deadlines of jobs can be specified according to the performance requirements. A deadline-driven resource provision mechanism was presented to support QoS-aware execution of scientific workloads in heterogeneous cloud environment by Vecchiola et al. [2012]. Malawski et al. [2012] address a resource management problem concerning IaaS project with cost budget and deadline constraints. The problem of minimizing the cloud operation cost by maximizing its energy efficiency while ensuring the application's QoS requirements is addresses by Gao

et al. [2013] later. Yang et al. [2013] apply a dynamic interference sensitivity detection methodology to preserve the performance of batch-analysis applications for collocation scenarios. Han et al. [2014] try to reduces the costs incurred by cloud users that using IaaS by utilizing adaptive scaling algorithms for cloud resources, which enable them to scale their applications only meets bottleneck. Poola et al. [2014] presented a robust scheduling algorithm with resource allocation policies that schedules workflow tasks on heterogeneous cloud resources while trying to minimize the total elapsed time (make span) and the cost. Singh and Chana [2015] categorize the cloud application workload on the basis of common patterns and then allocating the resource according to the generalized patterns before actual scheduling. Fernandez et al. [2014] proposed a system which selects a resource scaling plan according to both workload and customer requirements.

2.2.1.2 Resource Based

A theoretical problem formulation is developed for allocating multiple heterogeneous types of resources to competing cloud services and the proposed algorithms are compared through simulation experiments based on the Google Cluster Workload [Stillwell et al., 2012]. Xiao et al. [2013] introduce a new concept, "Skewness", to measure the unevenness in the multi-dimensional cloud resource utilization. They proposed a system to combine different types of workloads and improve the overall cloud resource utilization by minimizing Skewness [Mars and Tang, 2013]. Klein et al. [2014] introduce Brownout that uses a self-adaptation programming paradigm based on Control Theory to develop applications that can robustly withstand unpredictable resource performance without over-provisioning.

2.2.1.3 Bargaining Based

Lai et al. [2005] develop a cloud resource allocation system based on bargaining, which allows applications to differentiate the values of its jobs. While An et al. [2010] propose an alternative approach where applications are allowed to automatically negotiate resource leasing contracts with cloud providers. Similarly, Dastjerdi and Buyya [2012] propose a solution to automate the negotiation process in cloud environment. Zhang et al. [2011] try to address the question how to best match applications QoS requirement in order to maximize cloud provider revenue and cloud users satisfactions while minimizing energy cost in a single cloud provider scenario. Zaman and Grosu [2013] attempt to formulate the problem of resource allocation in clouds as a on-line auction problem.

2.2.1.4 Prediction Based

A resource allocation methodology is presented by Gmach et al. [2007], which relies on the ability to predict the cloud application's behaviour a priori while Gong et al. [2010] propose an alternative schema based on predictions of dynamic cloud resource run-time performance. Watson et al. [2010] study the probabilistic relationships between resource and application and apply basic laws of probability to their proposed model to investigate whether and how CPU utilization affects application performance. Shen et al. [2011] use on-line workload demand prediction without a priori assumptions on application behaviour to identifies the application's resource requirement, which attempt to avoid over-provisioning or over-loading of cloud resources. An algorithm is proposed by Li et al. [2012] to adjust the number of resource allocated to applications based on the updated information of their actual task executions. Islam et al. [2012] present a new resource measurement and provisioning solution based on prediction using Neural Network and Linear Regression to meet future workload demands while Vasic et al. [2012] serves a similar goal by classifying workload and reuses previous resource allocations decisions to minimize reallocation overheads. In Jiang et al. [2013]'s work, they attempt to make a trade-off between resource demand and service latency by automatically predict the number of application query requests .

2.2.1.5 Nature-inspired / Bio-inspired Based

Hegazy [1999] use the Genetic Algorithms (GAs) technique to search for near-optimum solution by taking both resource allocation and leveling heuristics into consideration. Hua et al. [2010] proposed an Ant Colony Optimization (ACO) based resource allocation algorithm to satisfy the property of cloud computing. A novel parallel Q-learning approach is presented by Barrett et al. [2013] to reduce the overhead introduced by determine optimal policies while learning on-line. Recently, a self-tuning fuzzy control (STFC) approach is extended to enable qualitative specification of elasticity rules for applications running on the cloud [Rao et al., 2013].

2.2.2 Job Scheduling

Once the resources are allocated to the cloud application, the RMS will perform scheduling decisions on the incoming jobs. Ideally, the job scheduler should have three desirable properties:

1. each job should receive sufficient resources to enable it to achieve *high and predictable performance*;

2. jobs should be tightly scheduled on available resources to achieve *high resource utilization*; and

3. scheduling overheads should be minimal to allow the scheduler to *scale to large clouds and high job arrival rates.*

With these three objectives in mind, job schedulers follow a diverse set of designs, which can be examined via their properties of throughput, latency, predictability, efficiency, overhead and failure rate. As mentioned in the former section, the optimal matching of jobs to suitable resources is an optimization problem, generally with NP-complete complexity. Heuristics are often applied as suboptimal algorithms to obtain relatively good solutions. Generally, one of two main strategies is pursued in heuristic solutions, static or dynamic. Static heuristics are suitable for the situation where the complete set of tasks is known prior to execution, while dynamic heuristics assess and perform the scheduling required when a task arrives. Comparative studies show that static heuristics outperform dynamic heuristics in most cases and from different perspectives [Blythe et al., 2005; Braun et al., 2001; Lopez et al., 2006; Wieczorek et al., 2005].

2.2.2.1 Static Heuristics

In static heuristics, jobs arrive simultaneously and the available resource schedules are updated after each task is scheduled. This scheduling assumes a precise knowledge of the timing information concerning the jobs which is difficult to obtain, but less runtime overhead is incurred when such is available.

Opportunistic Load Balancing (OLB) assigns each job, in arbitrary order, to the next resource that is expected to be available, regardless of the job's expected execution time on that resource [Freund and Siegel, 1993]. **OLB** tries to keep all resources as busy as possible. The main advantage of **OLB** is its simplicity. However, **OLB** results in very poor overall project completion time since it does not consider expected job execution times.

Minimum Execution Time (MET) assigns each job, in arbitrary order, to the resource with the shortest expected execution time for that job, regardless of that resource's availability [Freund and Siegel, 1993]. **MET** tries to schedule the job to the fastest/best resource for it - which can cause load imbalances across resources.

Minimum Completion Time (MCT) assigns each job, in arbitrary order, to the resource with the minimum expected completion time for that job [Freund and Siegel, 1993]. **MCT** tries to avoid the circumstances in which **OLB** and **MET** perform poorly.

Min-Min begins with the set U of all unscheduled jobs. The matrix of minimum completion times for each job in the set U is calculated. The job with the minimum completion time, overall, is selected and assigned to the corresponding resource. Finally, the scheduled job is removed from U, and the process repeats until all jobs are scheduled [Freund et al., 1998].

Max-Min is similar, but opposite, to the Min-Min heuristic. The job with the maximum completion time, overall, is scheduled to its corresponding resource [Freund et al., 1998].

Duplex is literally a combination of the **Min-Min** and **Max-Min** heuristics [Freund et al., 1998]. **Duplex** tries to switch between these schemes according to conditions which indicate which of them will perform well in present circumstances - but with negligible overhead.

Genetic Algorithm (GA) is a technique used for searching large solution spaces [Wang et al., 1997]. The characteristics of **GA** heuristics (in this context) include: a separation of the matching and the scheduling representations, independence of the chromosome structure from the details of the communication subsystem, and consideration of overlaps among all computations and communications tasks that are subject to job precedence constraints.

Simulated Annealing (SA) uses the same chromosomal representation as is used in **GA** [Coli and Palazzari, 1996] but considers only one possible solution for each job at a time. Since **SA** may accept a worse makespan, based on a probability, it finds poorer solutions than **GA**.

A* is a tree-based search heuristic beginning at a root node that is a null solution [Chow and Liu, 1991]. When the tree is grown, the nodes will represent partial scheduling (a subset of jobs is assigned to resources), and the leaves will represent final scheduling (all jobs are assigned to resources). **A*** aims at producing a schedule of minimum execution time when a leaf (complete scheduling) is reached.

2.2.2.2 Dynamic Heuristics

With dynamic heuristics, it is assumed that the timing information concerning the jobs and the resources is not known at runtime. For example, not all jobs arrive simultaneously and some resources go off-line at intervals. Dynamic heuristics can be used in two ways: in real-time mode and in batch mode.

In real-time mode, each job is scheduled only once and the scheduling result cannot be changed. This is suitable for the cases in which job arrival rate is low [Maheswaran

et al., 1999]. Some real-time scheduling algorithms are as follows: **Opportunistic Load Balancing (OLB)**, **Minimum Execution Time (MET)**, **Minimum Completion Time (MCT)**, **Simulated Annealing (SA)** and **K-Percent Best (KPB)**. **KPB** considers only a subset of resources while scheduling a job. The subset is formed by picking the k best resources, based on the execution times for the task. The purpose is to avoid allocating the current job a resource which might be more suitable for some yet-to-arrive jobs, so it outperforms others in most scenarios. The results of **MCT** are good, only slightly worse than those of **KPB**, owing to the lack of prediction for job heterogeneity [Maheswaran et al., 1999].

In batch mode, jobs are not scheduled as they arrive, instead they are collected and scheduled only at some predefined instances in time. Both **Min-Min** and **Max-Min** are batch scheduling algorithms.

2.2.2.3 More Heuristics Based On Objectives For Job Scheduling

Budget is the expected cost which the users will have to pay to rent the cloud resources. The goal of budget-based job scheduling is to finish all jobs as fast as possible at a given budget. The **Heterogeneous Earliest-Finish-Time (HEFT)** heuristic, proposed by Topcuoglu et al. [2002], selects the job with the highest upward rank value at each step and schedules the selected job to the resource, which minimizes its earliest finish time,using an insertion-based approach. Lin and Wu [2013] formulated the job scheduling problem in such a way as to minimize the workflow end-to-end delay under a user-specified budget constraint.The **Genetic Algorithm (GA)** approach presented by Yu and Buyya [2006] can be applied to search for time optimal solutions within budgetary constraints.

Deadline is the time limit for the execution of the jobs submitted to the user's cloud applications. The simplest solution to deadline-based job scheduling is the minimum critical path methodology. This only selects cheaper resources for non-critical jobs when the execution of critical jobs is not thereby influenced. If the makespan would terminate before the deadline, then there is a potential to further reduce cost by delaying the start of the makespan so that it finishes, at latest, at the deadline. In the **Deadline Early Tree (DET)** heuristic, all jobs are partitioned into different paths of an Early Tree [Yuan et al., 2009]. For critical jobs, the whole deadline period is segmented into time windows. For non-critical jobs, an iterative procedure is may be used to determine the time windows while keeping to the precedence constraints among the current jobs. Then a dynamic programming strategy is adopted to search for local optimal resources for jobs.

Reliability is the probability that the job can be completed as expected, which is a major performance issue in the presence of resource performance fluctuations. Lee and Zomaya [2010] investigated the effectiveness of rescheduling jobs in the cloud to increase the reliability of job completion time. Zhao et al. [2013] examined the problem of reliable jobs scheduling with less resource redundancy. Algorithms are proposed to avoid the "Chain effect" caused by uncertainties in relation to job execution time estimates, and relieve the impact of inaccuracy caused by poor estimations.

Load Balancing optimizes the resource usage in order to avoid overloading. Whenever certain resources are overloaded and remaining resources are under-loaded with jobs for processing, the load is balanced in order to achieve optimal resource utilization. **Load Balanced Improved Min-Min (LBIMM)** presented by Chen et al. [2013b] modified the basic **Min-Min** heuristic by improving the load balance to effectively reduce the execution time on each resource. LD and Krishna [2013] proposed an algorithm named **Honey Bee Behaviour inspired load balancing (HBB-LB)**, which aims to achieve well balanced load across virtual machines for maximizing throughput.

Multi-Objectives heuristics consider more than one objective in making scheduling decisions. Yu et al. [2005] proposed **Deadline-MDP** that minimizes execution cost while meeting the deadline for cost-based workflow. Wu et al. [2013] proposed a market-oriented hierarchical job scheduling strategy in the cloud based on users functional and non-functional QoS requirements. Liu et al. [2011b] adopts the use of the **Ant Colony Optimization (ACO)** algorithm to optimize cloud job scheduling with respect to various quality of service (QoS) requirements. Service Level Agreements (SLA) are introduced in their model and a SLA monitoring module is also implemented in order to monitor the operational state of cloud services.

2.2.3 Resource Management Systems in Industry

The outstanding performance of the current Apache Hadoop system [White, 2012] in big data processing scenarios has received the regard of many from the industry. The most popular resource management system for running Hadoop MapReduce applications on the cloud are **YARN** [Vavilapalli et al., 2013] and **MESOS** [Hindman et al., 2011]. In recent years, Apache Spark [Zaharia et al., 2010] has continued to attract attention in the big data world; it claims to run, when in memory only, up to 100x faster than Hadoop MapReduce, and 10x faster when on disk. Spark applications can run on the cloud under its own **Spark Standalone** mode [Zaharia et al., 2010], or connect with either **YARN** or **MESOS**. We will discuss these RMSs in more detail, in the following.

2.2.3.1 Apache Hadoop NextGen MapReduce (YARN)

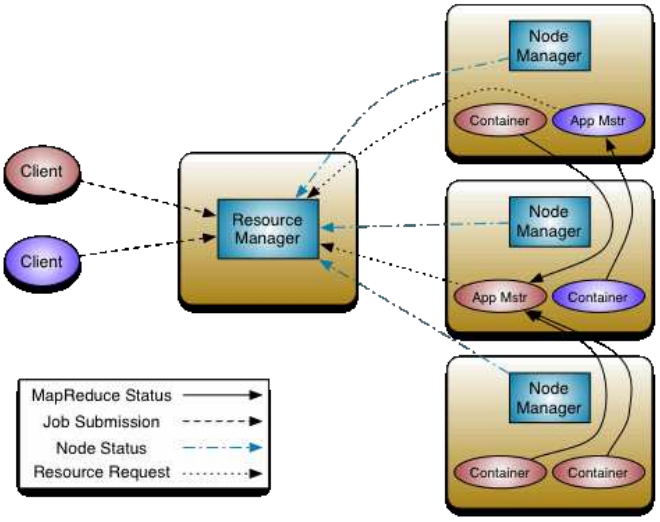

FIGURE 2.2: Apache YARN architecture [Vavilapalli et al., 2013]

YARN (Yet Another Resource Negotiator) re-architects the original MapReduce by splitting up resource management and job scheduling/monitoring into separate modules. A typical **YARN** cloud consists of a single *resource-manager* and multiple *node-managers*, as shown in Fig. 2.2. Applications submitted to **YARN** are run on *containers*, which are process abstractions that can run any user application. These *containers* are monitored by local *node-managers* and allocated by the *resource-manager*. The *node-manager* is responsible for reporting its containers' capacities (Memory and CPU Core limits) and the progress status of running applications to the *resource-manager*, periodically. The *resource-manager* is responsible for allocating *containers* to the various running applications subject to the familiar constraints of capacities, queues etc. After the *resource-manager* assigns *containers* to the applications, it performs its scheduling function based on the resource requirements of the application's map and reduce jobs. In the current **YARN** version, only Memory and the number of CPU Cores are considered as constraints during the job scheduling process. The default **YARN** schedulers include the **FIFO Scheduler**, the **Fair Scheduler** and the **Capacity Scheduler** [Zaharia, 2009].

FIFO Scheduler: The default Hadoop scheduler operates using a FIFO queue. After a job is partitioned into individual tasks (Map tasks and Reduce tasks), it is loaded into the queue and assigned to free cores as they become available on the *containers*. Typically, each job uses all the containers that are allocated to it, so jobs have to wait for their turn, which means the application can only run one job at a time.

Fair Scheduler: The Fair Scheduler [Zaharia, 2010] aims to give every job a fair share of cores over time. Applications may assign jobs to pools and each pool is allocated a guaranteed minimum number of CPU cores. Free cores in idle pools may be consumed by jobs in other active pools, while excess capacity within a pool is shared among jobs. The Fair Scheduler supports preemption, so if a pool has not received its fair share for a certain period of time, then the scheduler will kill jobs in pools having over capacity in order to give the CPU cores to the pool having under capacity. In addition, users may enforce the priority setting of certain pools. Jobs are therefore scheduled in an interleaved manner, based on their priority within their pools, the capacity of all the containers and the usage of other pools. In such a situation, shorter jobs are able to finish quickly while longer jobs are run at the same time.

Capacity Scheduler: The Capacity Scheduler [Raj et al., 2012] addresses a usage scenario where the number of applications is large, and there is a need to ensure fair allocation of resources among applications. The Capacity Scheduler allocates jobs, based on its application, to queues with configurable numbers of containers. Queues which contain jobs are given their configured capacity, while free capacity in a queue is shared among other queues. Overall, in this approach, all the containers are enforced to be shared among all the applications, rather than among jobs, as was the case in the Fair Scheduler.

2.2.3.2 Apache Mesos

Apache Mesos is another popular cloud resource manager which is capable of running MapReduce applications. Its architecture is quite similar to that of YARN, as shown in Fig. 2.3. In contrast to YARN, however, MESOS can run any kind of application rather than just MapReduce applications - which are specifically targeted by YARN. MESOS consists of a *master* process that manages *slave* daemons running on each resource, and schedules jobs on these *slaves*. Mesos delegates scheduling decisions to a pluggable scheduler module, so that applications can tailor the scheduler to their needs. So far, MESOS have implemented two scheduler modules: one performs fair sharing based on a generalization of Max-Min fairness for multiple resources and another implements strict priorities like YARN.

2.2.3.3 Apache Spark Standalone Mode

In addition to running on YARN and MESOS, Spark also provides a simple standalone mode. In standalone mode, each application is assigned a Spark *driver*, which is the process running the Spark context. This driver is responsible for converting the jobs

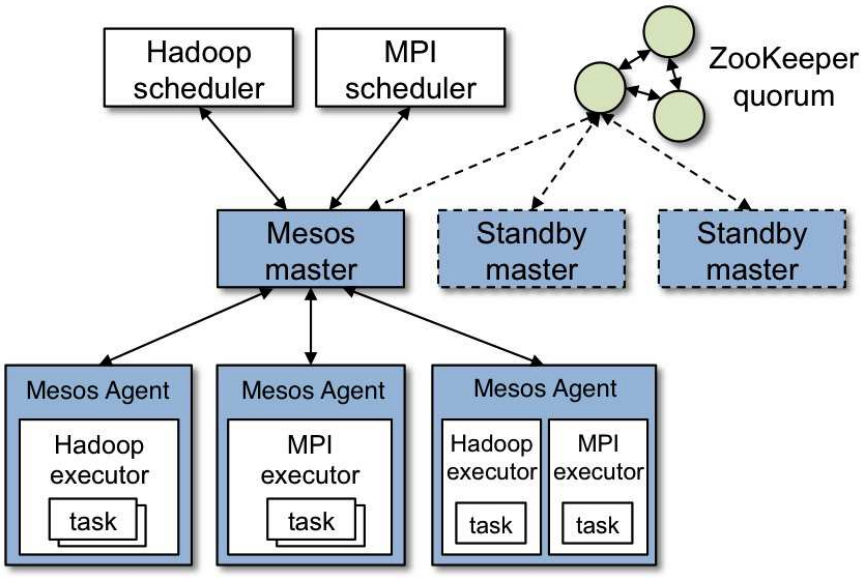

FIGURE 2.3: Apache MESOS architecture [Hindman et al., 2011]

to a directed graph of individual tasks to be executed on the *executors*. In the current version, Spark Standalone Mode has been implemented with two schedulers, a FIFO Scheduler and Fair Scheduler; this is similar to the scheduling policy used in YARN.

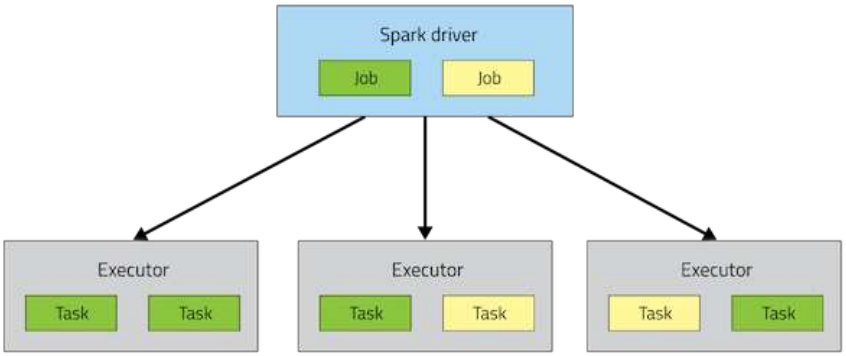

FIGURE 2.4: Apache Spark Standalone Mode architecture

2.2.4 Cloud Simulation Tools for Resource Management Research

To simulate the cloud environment in order to test resource allocation and job scheduling in different contexts, some prominent simulations tools are available. Among these, the most widely-used simulation tool is the **CloudSim** tool-kit which tests the execution time, costs and energy consumption involved in a cloud scenario by extending existing classes according to the requirements of an algorithm [Calheiros et al., 2009]. The **CloudSim** tool-kit can also provide important functionalities like application services, storage services, resource provisioning, simulate federated clouds, etc. The prominent simulation tools used for cloud resource management are as follows.

2.2.4.1 CloudSim

The primary objective of this **CloudSim** system is to provide a generalized, and extensible simulation framework that enables seamless modeling, simulation, and experimentation with emerging cloud computing infrastructures and application services [Buyya et al., 2009; Calheiros et al., 2009, 2011]. By using CloudSim, researchers and industry-based developers can focus on specific system design issues that they want to investigate, without getting embroiled in low level details related to Cloud-based infrastructures and services. In the past, there have been projects concerned with extending the power of **CloudSim** via different focuses, such as **CloudAnalyst** [Wickremasinghe et al., 2009], **RealCloudSim** [Agostinho et al., 2011], **WorkflowSim** [Chen and Deelman, 2012], **Cloud2Sim** [Kathiravelu and Veiga, 2014], **DynamicCloudSim** [Bux and Leser, 2015] etc.

2.2.4.2 GreenCloud

GreenCloud is a sophisticated packet-level simulator for energy-aware cloud computing data centres, with a focus on cloud communications [Liu et al., 2009]. It offers a detailed fine-grained modeling of the energy consumed by the data centre IT equipment, such as computing servers, network switches, and communication links.

2.2.4.3 ICanCloud

ICanCloud is a simulation platform aimed at modelling and simulating cloud computing systems, which is targeted at those users who deal closely with these kinds of systems [Núñez et al., 2012]. The main objective of **ICanCloud** is to predict the trade-offs between cost and performance of a given set of applications executed on a specific hardware configuration, and then to provide useful information about these costs.

2.2.4.4 Yarn Scheduler Load Simulator (SLS)

SLS can simulate large-scale Yarn clusters and application loads in a single machine [Apache Hadoop, 2013]. This simulator is invaluable in furthering Yarn systems by providing a tool by which researchers and developers can prototype new scheduler features and predict their behavior and performance with a reasonable amount of confidence, thereby aiding rapid innovation.

2.3 Complexities In Cloud Resource Management System

Resource management is the core functionality required for cloud systems. It affects the three basic criteria for system evaluation: performance, functionality, and cost. Inefficient resource management has a direct negative impact on performance and cost. It can also indirectly affect system functionality. Some features provided by the system may become overly expensive or ineffective due to poor performance.

A cloud computing infrastructure is a complex system with a large pool of shared resources, which run-time performance is highly dynamic and may be affected by external events beyond your control. Cloud resource management requires complex policies and decisions for multi-objective optimization. This is very challenging due to the complexity of the system, which makes it impossible to have accurate global status information. It is also subject to incessant and unpredictable interactions with the environment. In the cloud, where changes are frequent and unpredictable, considering complexity in resource management is of great interest due to the scale of the system, the large number of service requests, the large user population and the unpredictability of the load. However, many techniques are concentrated on static characteristics in terms of number of CPU cores, which rarely include QoS guarantees. Some techniques are based on unrealistic assumptions. For example, capacity allocation is viewed as an optimization problem, but under the assumption that cloud resources are running with unchanged performance.

Controlling the resources in cloud computing must take the complexity factors into account, rather than the static characteristics. Some of these complexity factors are as follows:

- **Heterogeneity**: Current cloud infrastructures are not yet very versatile, but heterogeneity is among the most important features which must be taken into account in any cloud system. With the development of virtualization technology, a single physical host can run multiple virtual machines (VMs) simultaneously. Nevertheless, this virtualization also brings about new challenges to resource scheduling in clouds due to the existence of multiple VMs which share the hardware resources (e.g. CPU, memory, I/O, network, etc.) of a physical machine. In such situations, it is difficult to accurately measure the actual performance of rented VMs. For example, in Amazon EC2, the provisioning of resources to virtual machines is based on compute units instead of fixed performance measures. Different host machines provide a different amount of computing power per provisioned compute unit, effectuating a heterogeneity among VM performance [Iosup et al., 2011]. This means, in the real world, that the cloud is never homogeneous but always heterogeneous.

- **Dynamicity**: Dynamic changes of resource performance at runtime is another important complexity factor inherent to cloud computing [Schad et al., 2010]. In the real world scenario, such dynamicity of resource performance may be caused by hardware/software failures, resource CPU overload, resource over- or under-provisioning, or application misbehaviours. A cloud resource is also affected by the amount of running jobs that are assigned to it and exhibited local activity; this is the origin of the complexity. Furthermore, sharing common underlying hardware infrastructures with other VMs will also increase the complexity level relating to resource dynamicity.

- **Uncertainty**: The vast majority of research efforts related to scheduling assume complete information about the state of cloud resources. However, in cloud computing, the ready time and the computing capacity of a resource are subject to considerable uncertainty during provisioning [Herroelen and Leus, 2005]. We argue that such uncertainty is the main difficulty with cloud computing, bringing additional challenges in terms of predicting the execution time of tasks, which is a crucial point for many scheduling algorithms. Resource states in cloud environments can change dramatically. Most of the time, it is impossible to get exact knowledge about a resource. It is hard to estimate the runtime of tasks accurately, improve prediction by historical data, perform prediction correction, undertake prediction fallback, etc. Imprecise execution prediction times leave the associated scheduling performance under considerable uncertainty.

2.4 Conclusion

In this chapter, I have discussed the related works on resource management when running MapReduce cloud-based applications. Cloud-based resource management has been a common area for research by many research communities over the past few years. However, much of the past research work has not considered the complex nature of the cloud environment and all the solutions used in the industry treat the cloud environment as something which is relatively simple. Effective cloud resource management helps to improve the utilization of resources and so reduce execution cost, execution time and execution variance and so has an effect on other QoS parameters like reliability, security, availability and scalability. To make optimal resource allocations, we need to take the complexity of cloud resources into account. However, the lack of information sharing between the cloud user and the cloud provider regarding cloud resources makes this problem more challenging.

As it stands, the challenges of resource allocation, such as the complexity of resources (e.g. heterogeneity, dynamicity and uncertainty) have not been resolved using the established RMSs which exist in the cloud environment. Thus, there is a need to make cloud applications efficient by dealing with these properties of the cloud environment. Recently, some of the research in the field has started to tackle the complexity problem in order to make their resource management solutions in the cloud environment more robust; most of this research has focused on the simulation approach. However, to our knowledge, none of the current cloud simulators are capable of modeling the run-time complexity of the cloud environment. To resolve this problem, in the next chapter, I present ComplexCloudSim (a cloud simulator extending the popular CloudSim toolkit) which is a tool intended to assist in the understanding of the role of complexity in cloud resource management systems.

Chapter 3

Implementation: ComplexCloudSim

The vast majority of the research efforts on cloud resource management assume the cloud to be homogeneous and that the cloud resource's performance is determined and predictable. However, in the real world, there are numerous types of complexity associated with cloud resources, etc.: heterogeneity, dynamicity and uncertainty. For heterogeneous cloud resources with highly dynamic changes in performance, the expected execution times in regard to different cloud resources play a critical role in making management decisions, and differences between the actual execution time and the estimated execution time may significantly affect the performance of the cloud and cause resource management systems to be less robust.

In spite of extensive research into complexity issues in different fields, ranging from computational biology to decision making in economics, the study of complexity in cloud resource management systems is limited. In this chapter, I tackle the research question: what is the role of complexity in QoS-aware cloud resource management systems? **ComplexCloudSim** is presented. This extends the popular simulation tool-kit, CloudSim, by modelling the complexity factors in the cloud, including heterogeneity of resource, dynamic changes of run-time performance and the uncertainty of task execution times. The comparison of four widely used heuristic cloud scheduling algorithms when given inaccurate execution time information is used to evaluate the impacts of complexity on performance within cloud environments. Furthermore, I apply a damage spreading analysis, which is one of the available complex system analysis methods, to the system and to the simulations to show that the system reveals sensitivity to initial conditions in some parameter regions. Finally, I will discuss how small damage spreads throughout

the system in the region and discuss also research into the potential ways to avoid such chaotic behaviour and make the system more robust.

3.1 CloudSim : A Toolkit For Modelling And Simulation Of Cloud Environments

CloudSim is a popular framework for simulating resource scheduling on Cloud Computing infrastructures. When introducing CloudSim, it is important to mention the main entities/concepts it is based on, in terms of terminology:

- **Data centre** acts as Cloud Provider which contains a set of physical hosts that can either be homogeneous or heterogeneous with respect to their hardware configurations (CPUs, Memory, Bandwidth and Storage). This is responsible for resource provision to the cloud users.
- **Host** is a physical machine characterized by a list of CPUs (and their types), also the amount of memory they have, their storage and allocated bandwidth. A host is responsible for VMs management according to a specified VM allocation policy.
- **VM** stands for Virtual Machine. A VM is managed and hosted by a Cloud Host component.
- **Cloudlet** represents a job that is submitted by the Cloud User to run on the cloud. A job is characterized by length (millions of instructions), resource requirement (the number of cores and the amount of memory required for the job to be performed), dependencies and type (MapReduce-like jobs contain map tasks and reduce tasks).
- **Broker** is responsible for mediating negotiations between Cloud Users and Cloud Providers. A broker acts on behalf of the Cloud User to discover suitable resources which can be obtained from the Cloud Provider and undertakes online negotiations for the allocation of resources that can meet the user application's QoS needs. It then sends the cloudlets for scheduling to the available resources under specified scheduling policies.
- **CloudletScheduler** is responsible for determining the share of processing power among Cloudlets on available resources; This scheduler can be implemented with different scheduling policies.

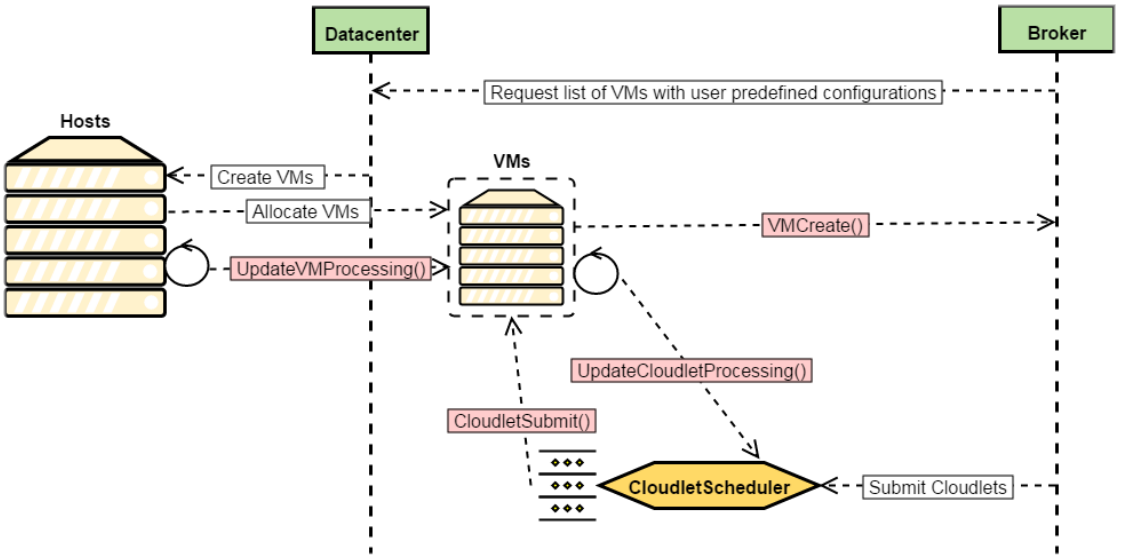

FIGURE 3.1: CloudSim : Simulation Flow Chart

In CloudSim and most of its extensions [Bux and Leser, 2013; Chen and Deelman, 2012; Garg and Buyya, 2011], the computational capabilities of hosts and VMs are measured by MIPS (million instructions per second per core). The MIPS measurement plays a major role throughout a CloudSim simulation. CloudSim assumes provisioned virtual machines to be predictable and stable in their performance. VMs are provided with guaranteed performance which is characterised as a fixed amount of MIPS and such performance is never changed during a simulation, as shown in Fig. 3.1. On actual cloud environments like Amazon EC2, these assumptions do not hold. Although most Cloud Providers guarantee a certain core speed for each provisioned VM, the actual performance of a given VM is subject to the underlying physical hardware as well as to the runtime CPU utilization of the host that the VM is assigned to. Thus, such incorrect assumptions mean that CloudSim fails to simulate well the complexity of the cloud environment.

3.2 ComplexCloudSim : Modelling And Simulate The Complexity In The Cloud

We explain in this section how complexity can affect cloud simulations through a study of four popular cloud scheduling algorithms and a motivational example. Then we present the proposed ComplexCloudSim that incorporates cloud complexity into the original CloudSim.

3.2.1 Cloud Scheduling Algorithms

In general, a scheduling algorithm is implemented in a cloud scheduler that will be permanently running as follows: receive new incoming jobs, check for available resources, select the appropriate resources according to feasibility (jobs' requirements to resources) and performance (estimated time to be completed) criteria and produce a planning of jobs (making the decision about job ordering and priorities) to selected resources.

Usually the following terminology - in Table 3.1 - is employed in relation to scheduling in clouds.

TABLE 3.1: Terminology For Scheduling In Cloud Computing

Name	Description
QoS	Quality of the service
$MIPS$	Million instructions per second (CPU processing speed)
L_t	Length of task measured in million of instructions
ETC	Estimated time to compute
ERT	Estimated ready time of resource
MCT	Minimum completion time matrix
$Makespan$	Project completion time
M_e	Estimated makespan
M_a	Actual makespan

Four widely utilised heuristic scheduling algorithms are used for performance evaluation purposes in relation to simulations of cloud-based complexity in this thesis. Definitions of these four heuristics are provided below.

- **FCFS (First Come, First Served)**: Tasks are executed according to the sequence in which they were submitted. The task which arrives first will be scheduled on the available resource first, as soon as it is submitted; it will then be removed from the queue.
- **Round Robin**: Schedules the first task on the first resource, the second task on the second resource and so on, cycling through all the available resources.
- **MinMin**: All the tasks in a job will first be ordered according to their length (of execution). The task with the shortest length will be scheduled first on the

available resource for which the completion time will be minimum - and then removed from the queue.

- **MaxMin**: All the tasks in a job will first be ordered according to their length (of execution time). The task with the minimum length will be scheduled first on the available resource - for which the completion time is maximum - and then removed from the queue.

3.2.2 Motivational Example

In this section, I will demonstrate how the complexity of resources impacts the robustness of a scheduler. Consider the case of ten independent jobs that need to be scheduled in a homogeneous cloud with three VMs, with specifications as shown in Table 3.2 and Table 3.3. To simplify the complexity of scheduling, we assume the length of jobs, measured by million instructions (MIs), is known and fixed and other performance related features of the cloud have no impact on the actual completion time of the jobs - such as memory consumption, network bandwidth, disk I/O.

TABLE 3.2: Jobs Specifications

Job Number	Number of Tasks	Task Length (MIs)
1	3	100
2	2	80
3	8	70
4	4	100
5	3	80
6	3	20
7	2	50
8	6	60
9	2	90
10	4	150

Table 3.3: VMs Specifications

VMs	Core#	$MIPS_{request}$	$MIPS_{provision}$
VM1 (4 Cores)	1	10	9
	2	10	9
	3	10	9
	4	10	9
VM2 (4 Cores)	5	10	10
	6	10	10
	7	10	10
	8	10	10
VM3 (4 Cores)	9	10	11
	10	10	11
	11	10	11
	12	10	11
Total 3 VMs	12 Cores	120	120

In this example, I use the Min-Min heuristic to schedule these independent jobs; this is a simple and efficient algorithm that produces, often, a better schedule (that minimizes the total completion time of jobs) than other algorithms in the literature. The pseudo code of the Min-Min algorithm is shown in Algorithm 1.

Algorithm 1 Min-Min Scheduling algorithm

```
1: Require: A set of jobs with n tasks, m different cores, MCT matrix
2: procedure MinMin Scheduling Algorithm
3:     A list of jobs L_j in queue
4:     A list of available cores L_c
5:     while (List L_j is not empty) do
6:         {For each job in the list L_j
7:         if (The number of avaliable cores meets the job's requirement) then
8:             {find the core that gives the minimum ETC
9:             and Update MCT matrix}
10:        From the MCT matrix, find the job with the minimum ETC
11:        Remove the job from the job list L_j
12:        Schedule the job's tasks to the match cores
13:        Remove the number of match cores from available cores list L_c}
```

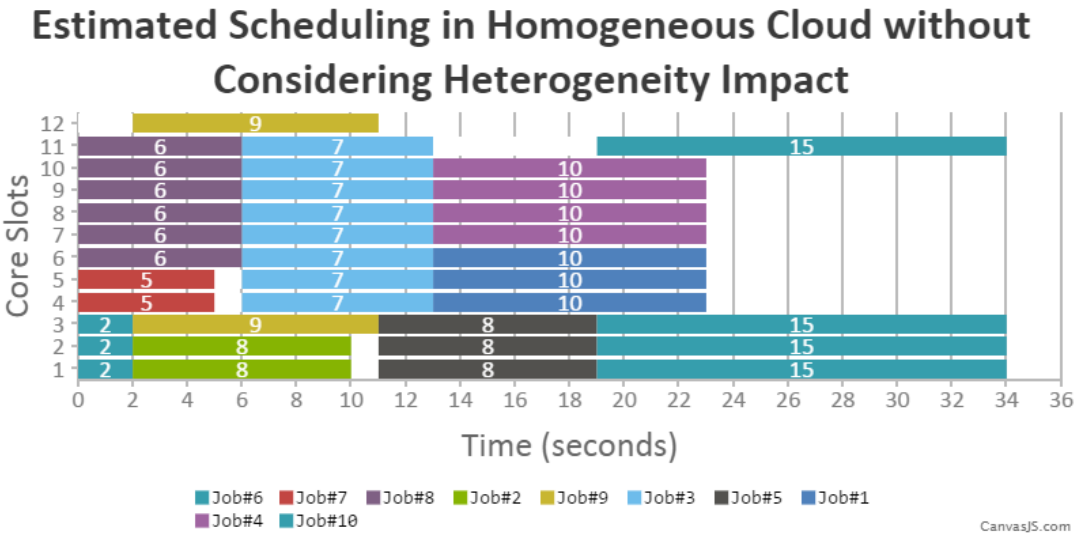

FIGURE 3.2: Motivational Example : Estimated Scheduling Plan

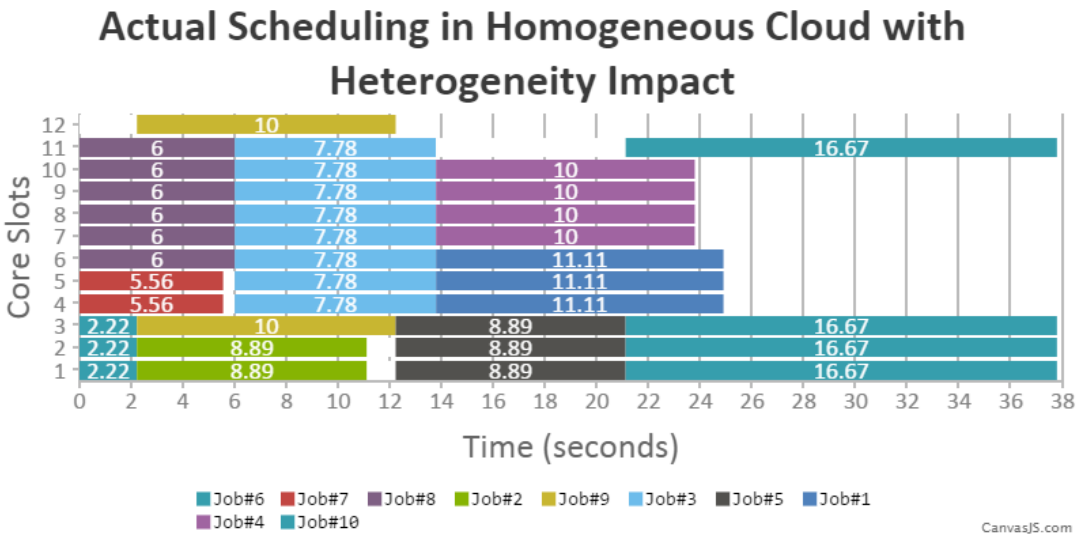

FIGURE 3.3: Motivational Example : Actual Scheduling Plan

As we can see from the difference between the estimated scheduling plan in Fig. 3.2 and the actual scheduling plan in Fig. 3.3, the complexity of resources have a great impact on the job's QoS. In this simple example, the complexity factor of resources is shown to degrade the robustness of scheduling algorithms, i.e. the average job makespan and the total workload runtime in this example, as shown on Table 3.4. Therefore, in the following sections, we will investigate how different degrees of complexity impact such robustness and how different scheduling heuristics perform under the complex cloud environment.

TABLE 3.4: Jobs Completion Details

Job Number	M_e	M_a	Makespan Degradation
1	23s	24.89s	1.89s
2	10s	11.11s	1.11s
3	13s	13.78s	0.78s
4	23s	23.78s	0.78s
5	19s	21.11s	2.11s
6	2s	2.22s	0.22s
7	5s	5.56s	0.56s
8	6s	6s	0s
9	11s	12.22s	1.22s
10	34s	37.78s	3.78s (11%)

3.2.3 The Implementation For Introducing Complexity

Although the simulator provides an approximation, faster and simpler simulation of application execution in the cloud, there are still many researchers who believe that these results cannot always be generalized for complex cloud environments. As I have discussed at the previous section 2, the performance of cloud scheduling is subject to different complexity factors relating to cloud resources: heterogeneity, dynamicity and uncertainty. However, existing simulators fall short in their modelling of such complexity factors common in the cloud environment. Although some simulators, such as DynamicCloudSim [Bux and Leser, 2015], offer users the capability to simulate the Cloud heterogeneity by introduce noisiness in dozens parameters. Still, it is difficult, or sometimes even impossible to determine appropriate values for all these parameters because they are usually Cloud or application-dependent. In ComplexCloudSim, we propose a new model that simplifies the simulation setup and reduces the bias between the behaviour of simulation and real Cloud environments based on only one parameter, Cloud errors. It represents the errors produced by inaccurate estimation of the cloud or application states. The injected Cloud error causes instability in job's execution time, which improves the accuracy, represented the trueness (i.e. closeness of the true mean value) and the precision (i.e. closeness of corresponding standard deviation) of the simulation, as defined in ISO-5725 standard [ISO, 1994].

In the remainder of this section, I will describe, in detail, how ComplexCloudSim attempts to capture the three common complexity factors (heterogeneity, dynamicity and uncertainty) by injecting Cloud errors in the simulation.

3.2.3.1 Cloud Error Produced by the Heterogeneity of VMs Provision

Algorithm 2 Heterogeneity Ratio for VMs Provision

1: **Require:** VMs MIPS configuration, $MIPS_{request}$
2: **Require:** Heterogeneity Ratio, $0 \leq Ratio_{heterogeneity} \leq 1$
3: **Require:** Cloud Error, randomly generated by a Mean Error of 0 and Standard Deviation of the value of heterogeneity ratio under normal distribution
4: **procedure** VMCreate($MIPS_{request}$,$Ratio_{heterogeneity}$)
5: **if** ($Ratio_{heterogeneity} > 0$) **then**
6: $\{MIPS_{provision} = MIPS_{request} * (1 \pm CloudError)\}$
7: **else**$\{MIPS_{provision} = MIPS_{request}\}$
8: VM Provision($MIPS_{provision}$)

In a similar way to the situation with a real-world Cloud Provider, the performance of the provisioning VMs is not guaranteed in ComplexCloudSim. Hence, VMs of equal configuration are likely to have different core performances characterised by the random changes of request MIPS during provision - unlike the guaranteed fixed MIPS provision of CloudSim. In ComplexCloudSim, we allocate MIPS to the VMs when they are created, with an injection of random cloud error by setting the *Heterogeneity Ratio*, as we can see from Algorithm 2. In this case, the heterogeneity ratio is the standard deviation for random generated cloud errors caused by the heterogeneity performance of provisioning VM.

3.2.3.2 Cloud Error Produced by the Dynamic Changes of VM performance at Runtime

The idea that there are dynamic changes to performance at runtime, due to the sharing of common resources with other VMs and users, is also an important concept relating to the complexity inherent to Cloud scheduling. In CloudSim, the VM performance is kept to a fixed number of MIPS during simulation. In ComplexCloudSim, we periodically, every second, change the VM's runtime MIPS by injecting random cloud errors by setting the Dynamicity Ratio and the host's current CPU utilization, as shown in Algorithm 3. In this case, the dynamic ratio is the standard deviation for random generated cloud errors caused by dynamic Changes of VM performance at runtime.

Algorithm 3 Dynamicity Ratio for Changes of VM performance at Runtime

1: **Require:** Host's CPU Utilization, U_{host}
2: **Require:** Dynamicity Ratio, $0 \leq Ratio_{dynamicity} \leq 1$
3: **Require:** Cloud Error, randomly generated by a Mean Error of 0 and Standard Deviation of the value of Dynamicity Ratio under normal distribution
4: **procedure** UPDATEMIPS(U_{host},$Ratio_{dynamicity}$) EVERY SECOND
5: **if** ($Ratio_{dynamicity} > 0$) **then**
6: $\{MIPS_{runtime} = MIPS_{provision} * (1 - U_{host}) * (1 \pm CloudError)\}$
7: **else** $MIPS_{runtime} = MIPS_{provision}$

3.2.3.3 Cloud Error Produced by the Uncertainty of VM Performance Estimation with Incomplete Information

Accurate resource performance prediction is hard or even impossible to achieve in actual complex cloud environments. CloudSim assumes that full information can be obtained and that such information is always correct for the purposes of performance prediction; this is not feasible in real world scenarios. Thus, we introduce the cloud error, by setting the Uncertainty Ratio, to reflect the estimation error of prediction due to incomplete information, which is used by several scheduling algorithms when making scheduling decisions (e.g. MinMin, MaxMin). In ComplexCloudSim, we inject the random cloud error into all the processes which need to perform performance prediction, according to the algorithm 4. In this case, the uncertainty ratio represents the standard deviation of random generated cloud errors in the uncertain prediction under incomplete information.

Algorithm 4 Uncertainty Ratio for VM Performance Estimation with Inaccurate Information in Scheduling

1: **Require:** Estimated VM performance, $MIPS_{estimate}$
2: **Require:** Uncertainty Ratio, $0 \leq Ratio_{Uncertainty} \leq 1$
3: **Require:** Cloud Error, randomly generated by a Mean Error of 0 and Standard Deviation of the value of Uncertainty Ratio under normal distribution
4: **procedure** PREDICTMIPS($MIPS_{estimate}$,$Ratio_{Uncertainty}$)
5: **if** ($Ratio_{Uncertainty} > 0$) **then**
6: $\{MIPS_{actual} = MIPS_{estimate} * (1 \pm CloudError)\}$
7: **else** $MIPS_{actual} = MIPS_{estimate}$

3.3 Complexity Simulation: Comparison of Four Heuristics Cloud Scheduling Algorithms

To showcase a possible application of ComplexCloudSim, we simulated the execution of a computationally intensive workload (the Montage workflow) using four different heuristic cloud scheduling algorithms and various degrees of complexity in the Cloud resources. We expected the schedulers to differ in their robustness in relation to complexity, and that this should be reflected in diverging workflow execution times. In this section, we outline the experimental setup and evaluate the impacts of resource complexity on Cloud scheduling systems.

3.3.1 Experiment Setup

To evaluate the robustness degradation caused by resource complexity, the following approach was used to simulate the scheduling system. For this experiment, we used a Montage workflow which comes with CloudSim; this consists of 1000 jobs with groups of random numbers of sub-tasks. For simplicity, we used a global variable, the degree of complexity, to configure the ratios of Heterogeneity, Dynamicity and Uncertainty all at the same time. For each configuration, the Montage workflow was executed 100 times on five virtual machines and the statistical results in terms of workflow runtimes were generated. In the course of the experiments, we incrementally raised the degree of complexity imposed by ComplexCloudSim, and by this means we measured the complexity's impacts on the QoS performance of cloud scheduling systems. For a comparison of ComplexCloudSim to the original Cloudsim, we conducted a baseline simulation which ran without taking into account any complexity factors; this was executed 100 times as well. As we expected, the workflow runtime for the same workflow under four scheduling algorithms was determined with zero variance in the original CloudSim, which is shown in Table 3.5.

TABLE 3.5: Baseline Simulation Result with Original CloudSim

Scheduling Algorithms	FCFS	RR	MinMin	MaxMin
Average Runtime (Minutes)	2862	2865	2864	2862

3.3.2 Experiment Result

Here, we compare the impacts on robustness which come about when using different scheduling algorithms and different degrees of resource complexity. The results of the experiment outlined above are displayed in Fig. 3.4 and 3.5. For all the experiments,

average runtimes of the Montage workflow between 3,220 and 3,505 minutes were observed and have been displayed in Fig. 3.4. This shows around 13% - 23% runtime degradation compared with the performance baseline. Apparently, the complexity factors provided by ComplexCloudSim have a considerable impact on the QoS of cloud scheduling system.

We also find that the average runtime degradation does not change directly in line with the increase in the degree of complexity. However, the increase in the standard deviation for workflow runtime is proportional to the increase in the degree of complexity with range from 20% to 120%, as shown on Fig. 3.5. Obviously, the growth in the standard deviation leads to less reliable scheduling performances. Thus, the reliability of the cloud scheduling system depends on the complexity of the resources.

In regard to both the average and the standard deviation of workflow runtime, the experimental results show that the MinMin scheduling algorithm is least impacted by the complexity factor. This means that MinMin generates more robust schedules in a complex cloud environment. So the overall performance of MinMin is better than other three heuristics, which confirm similar findings in earlier research.

Evidently, ComplexCloudSim can simulate the effect of complex resources. Since complexity is commonly encountered in cloud environments, this is very desirable property which will continue to be important going forward and has not been sufficiently supported by other cloud simulators.

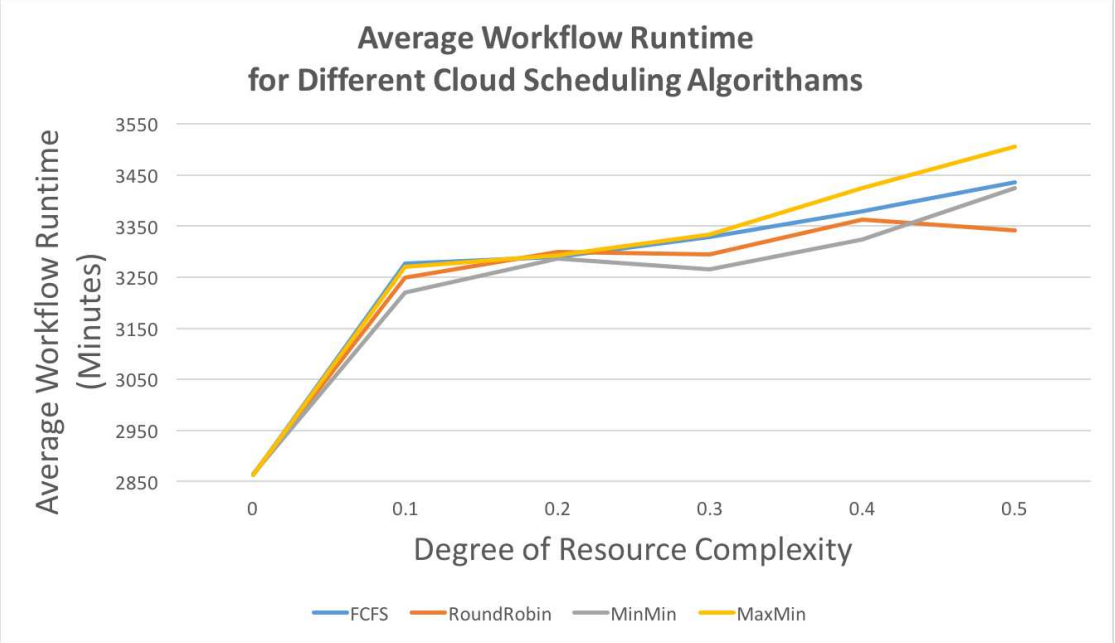

FIGURE 3.4: Complexity Simulation: Average Workflow Runtime

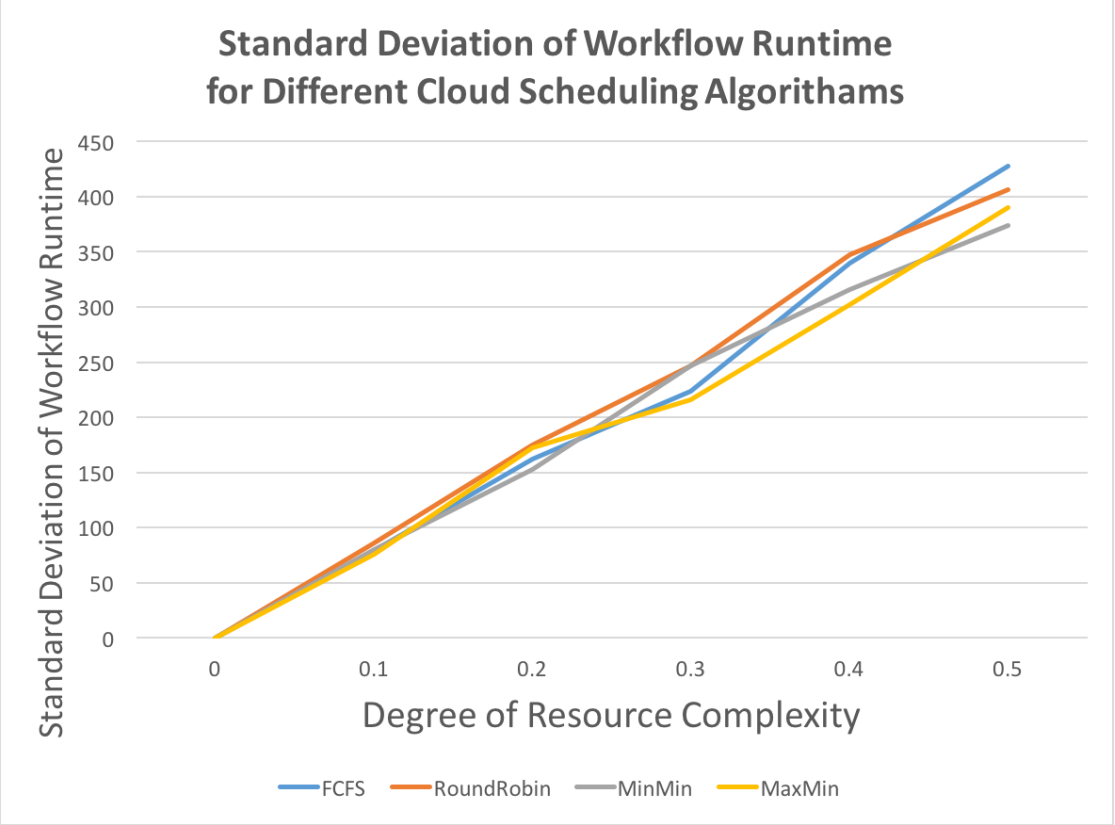

FIGURE 3.5: Complexity Simulation: Standard Deviation of Workflow Runtime

3.4 Damage Spreading Evaluation: Chaotic Behaviour in Cloud Scheduling

Damage Spreading [Kauffman, 1969] is a tool originally developed to study biologically motivated complex systems, and it appears in the literature on various research areas, including complex network models, as a way to observe the complex behaviour of systems. It investigates the evolution of slightly different configurations of variables in complex systems which are subjected to the same number sequence. Knowledge of whether or not a small perturbation ("damage" to the conditions) added to the variables spreads or stays at the same level (even disappears) can help us to investigate the robustness of a system in relation to disturbance [Ikeda, 2012].

"Initial damage" here is defined as a slight change in the degree of resource complexity, $C_{complexity}$, and the number of VMs, C_{vm}, to run the same workload. We added small changes, $C_{complexity} = 0.1$ and $C_{vm} = 1$, to a simulation step by step - the simulation is one which was executed 100 times with the same workload. Then we investigated whether the changes had spread or not in relation to two important QoS determinants of the scheduling processes - the average and standard deviation of workflow runtime.

To evaluate the spread of the damage, we defined damage as $D_{average}$ (difference in average workflow runtime $R_{average}$) and D_{std} (difference in workflow runtime Standard Deviation R_{std}) existing between two simulation results; these were calculated as shown in Formula 3.1 and 3.2, where $i \in [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$ refers to the number of VMs and $j \in [0.1, 0.2, 0.3, 0.4, 0.5]$ refers to the degree of complexity.

$$D_{average}(i,j) = R_{average}(i + C_{vm}, j) - R_{average}(i,j) \quad (3.1)$$

$$D_{std}(i,j) = R_{std}(i,j + C_{complexity}) - R_{std}(i,j) \quad (3.2)$$

The results of $D_{average}$ and D_{std} are shown in Fig. 3.6 and 3.7 respectively.

As we can see from Fig. 3.6, for a number of VMs of $i < 10$, the changes in $D_{average}$ for different degrees of complexity is relatively small; in this region, the damage is not spread and the damage stays at the low, initial, level.

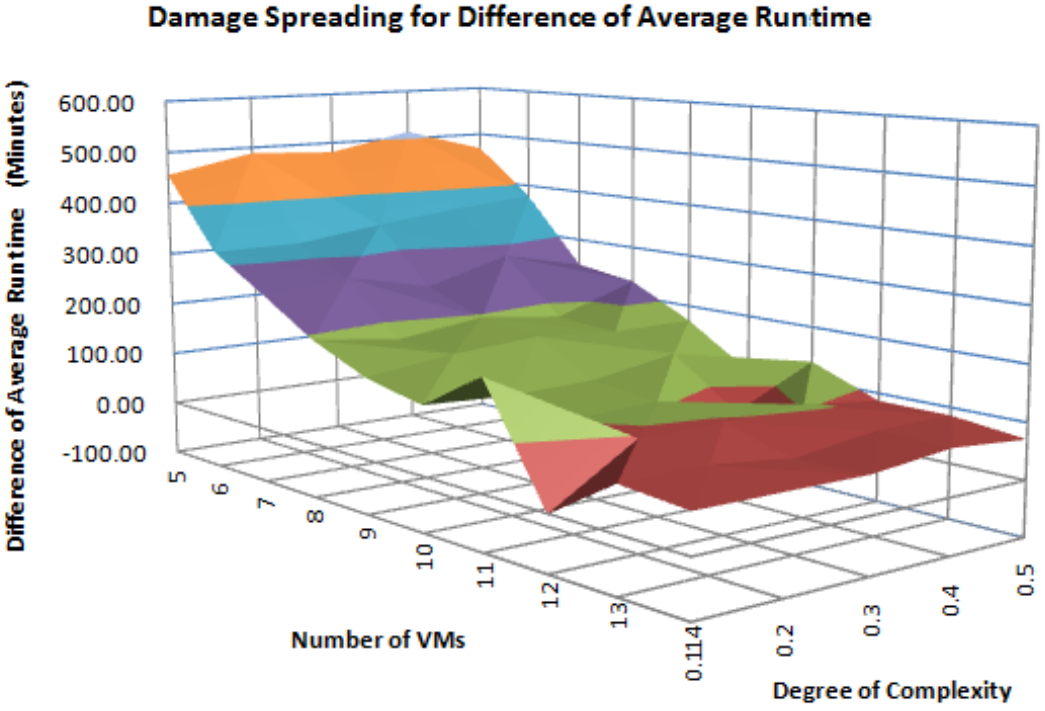

FIGURE 3.6: Damage Spreading Evaluation: $D_{average}$

From Fig. 3.7, for a number of VMs of $i < 9$, the changes in D_{std} for different degrees of complexity become highly unstable, but the situation becomes relatively better as the number of VMs is increased, when $i > 9$.

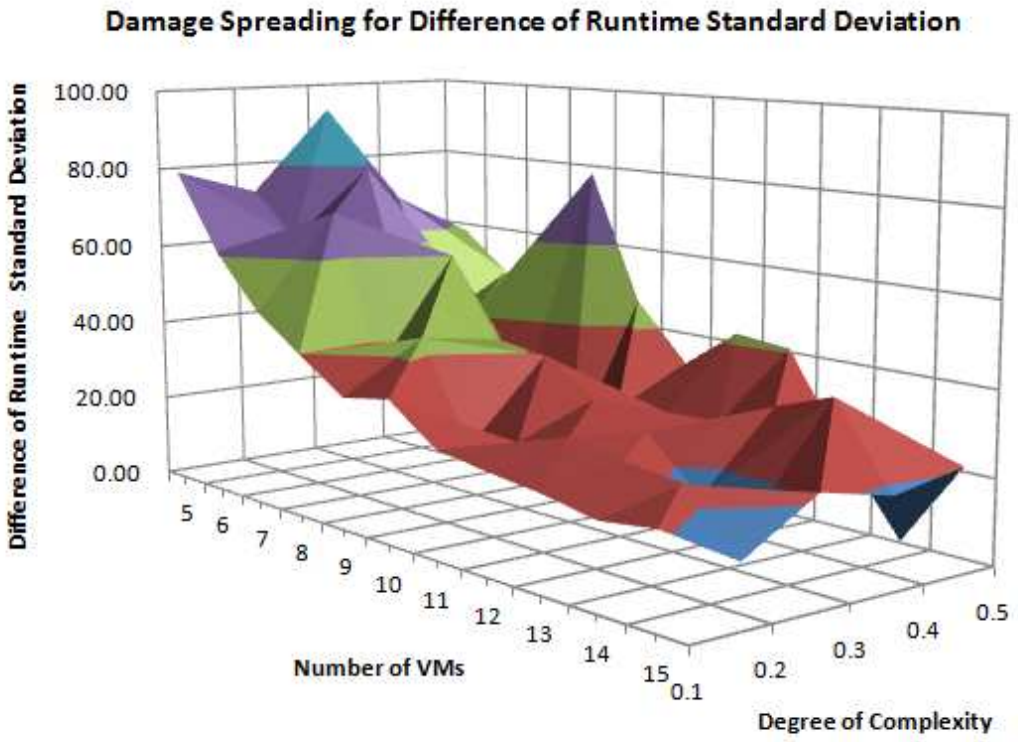

FIGURE 3.7: Damage Spreading Evaluation: : D_{std}

Then, we analysed the relation between number of increased VMs i and the spreading damage, using the standard deviation of $D_{average}$ and D_{std}. We defined the standard deviation of $D_{average}(i)$ as $\sigma_{average}(i)$, and the standard deviation of $D_{std}(i)$ as $\sigma_{std}(i)$. Hence we calculated the mean value $Mean(\sigma_{average})$ and the $Mean(\sigma_{std})$ of all $\sigma_{average}$ and σ_{std}, as shown on Table 3.6 and 3.7.

Now, we classify the state of the region loosely using such mean values. We understand the region that $\sigma_{average}(i) \leq Mean(\sigma_{average})$ or $\sigma_{std} \leq Mean(\sigma_{std})$ as "Stable Regions". In this region, a stable correlation between initial damage and spreading damage is maintained; the increase in the number of VMs will result in reliable improvements to QoS, which means the scheduling system is running relatively robustly against the changes in degree of complexity. We also understand that $\sigma_{average}(i) > Mean(\sigma_{average})$ or $\sigma_{std} > Mean(\sigma_{std})$ as "Chaotic Regions" [Boccaletti et al., 2000], as highlighted by the red colouring in Table 3.6 and 3.7 . In this region, small disturbances may spread throughout the scheduling system and the performance may readily be changed totally in response to the degree of complexity experienced, which means it is hard to guarantee QoS to an increased of number of VMs. The understanding of when the scheduling

TABLE 3.6: Relation Between Number of VMs and $D_{average}$

	$D_{average}(i)$					
	Degree of Complexity					Mean($\sigma_{average}$)=23
(i) VMs	0.1	0.2	0.3	0.4	0.5	$\sigma_{average}(i)$
5	456	489	481	514	469	22
6	320	322	344	363	377	25
7	258	271	237	282	248	18
8	193	174	196	178	231	23
9	148	168	180	169	171	12
10	124	117	122	149	94	19
11	198	101	108	64	135	50
12	-1	96	98	104	86	44
13	80	81	65	83	86	8
14	69	68	67	83	71	7

TABLE 3.7: Relation Between Number of VMs and D_{std}

	$D_{std}(i)$					
	Degree of Complexity					Mean(σ_{std})=24
(i) VMs	0.1	0.2	0.3	0.4	0.5	$\sigma_{std}(i)$
5	58	69	94	73	80	49
6	48	37	79	63	61	38
7	42	43	39	71	48	31
8	78	23	60	34	40	30
9	46	9	41	44	32	21
10	32	23	39	20	34	18
11	42	25	31	24	26	18
12	41	26	26	28	24	17
13	19	32	15	26	22	13
14	0	37	15	24	20	11
14	21	18	22	11	22	11

system is in a “stable region” and a “chaotic region” provides us an important guideline for making the decisions necessary to achieve more robust scheduling. For example, because of the results obtained by the simulation with ComplexCloudSim, in a real world situation we might run a similar workload with more than 9 VMs but we could avoid choosing 11 or 12 VMs in order to satisfy the QoS requirement.

3.5 Conclusion

To obtain the results presented in this section, ComplexCloudSim was implemented as an extension to CloudSim which evaluates scheduling under complex cloud environment. The resource complexity module (Heterogeneity, Dynamicity and Uncertainty) has been designed and implemented with the primary goal of providing a useful tool for validating and testing the robustness of cloud scheduling algorithms. The evaluation results of four cloud scheduling algorithms show that ComplexCloudSim is capable of simulating various complexity factors for cloud scheduling systems and was able to replicate the known strengths and shortcoming of these algorithms. Then, through the simulation, a region in the complex cloud scheduling situation was found in which small damage was converged, a "stable region", and another region was found in which the damage spread, a "chaotic region". I have a keen interest in finding "chaotic behaviour" within cloud scheduling systems because the presence of regions with chaotic behaviours means that we cannot, even in principle, predict the future as it relates to job scheduling within the cloud. Such findings may explain why most of the scheduling algorithms mooted that depend on prediction are difficult to implement effectively in the real world production environment, where complexity exists everywhere. In a complex production system like the Cloud, we usually will not know what the precise completion time of tasks will be even where we do know their precise theoretical processing time in advance. Hence, if a scheduling system wants to plan the production schedule more robustly, it has to judge whether it is in "Stable Region" or "Chaotic Region" first. Then, if the system is in chaos, it has to, for example, try to find VMs with sufficient resources to meet the application's QoS requirement.

Even through ComplexCloudSim is able to model some kinds of complexity factors, still, it is not able to cover all the situations which can happen in real world clouds. However, the finding there is "chaotic behaviour" in Cloud scheduling systems motivates new efforts to develop robust QoS-aware scheduling algorithms. For further work, more detailed analysis is needed to understand "chaotic behaviour" related to cloud scheduling systems and the mechanisms of damage spreading which come with it. Such chaotic behaviour should also be studied as it exists in real world cloud systems. Indeed, I believe that my work here represents a step towards many fruitful research topics.

Chapter 4

Complexity Management: Entropy-Based Cloud Resource Allocation and Job Scheduling

In cloud resource management systems, complexity limits the system's ability to adequately satisfy the QoS requirements of applications such as cost budgets, average job runtimes and reliability. Uncertainty, variety, diversity, numerousness etc. are some of the complexity factors which lead to the variation between the expected performance and the actual running performance of applications. In this chapter, after defining the complexity involved clearly, we classify this complexity into two general types: *Global System Complexity* and *Local Resource Complexity*. In order to manage complexity, an Entropy-based model is proposed which covers the identifying, measuring, analysing and controlling (reducing and avoiding) of complexity.

4.1 Complexity In Cloud Resource Management System

4.1.1 Definition And Classification

At the present time, the notion of complex systems has not been precisely delineated. However, although the idea of complexity is somewhat fuzzy and differs from author to author, there are some typical properties that can be seen to be shared by many complex systems.

- *Complex systems are made up of several non-linear components*

A cloud resource management system's resources serve as the cloud's basic components. These resources are non-linear. During run-time, the performance of the resource is highly dynamic and is influenced by the running jobs. Non-linearity is a condition that is needed for chaos. Furthermore, almost every system having a phase space with three or more dimensions can be considered chaotic in a certain part of that phase space [Larsen-Freeman, 1997].

- *A complex system's components are interdependent*

  The cloud's resources indirectly interact with each other via the resource management system. The state of the resources depends on other resources and is affected by the state of the other resources as well.

- *A complex system possesses a structure spanning several scales*

  Take the example of a typical cloud resource management system:

  - Scale 1: applications; resource management; resources ...
  - Scale 2: jobs, sub-tasks; resource allocation, job scheduling; hardware, software ...
  - Scale 3: functions, parameters, variables, requirements; constraints, objects; CPU, memory, storage, operating system ...
  - More scales : ...

  At every scale we find a structure. This is an essential and radically new (as in, newly discovered) aspect of complex systems and it leads to a fourth property...

- *A complex system can handle emerging behaviour*

  Emergence takes place when the focus of attention is shifted from one scale to another coarser scale above it. Observed at a specific scale, a certain behaviour is considered emergent if one cannot understand it after studying it separately and one by one. Each of this scale's components may also be a complex system that comprises finer scale. Therefore, the emerging behaviour is a novel phenomenon that is special to the scale being studied. Moreover, it is a result of the global interaction between that scale's components [Larsen-Freeman, 1997]. For instance, a computer has the ability to run a program, which is the highest scale's emerging behaviour. If the study is only focused on lower scale components like the transistor, wire, or power, one will never get an understanding of how the computer runs the program.

- *Complexity involves an interaction between chaos and order*

It has been said that many complex systems do not always display chaos at all times. In other words, they display chaos for some of the control parameter's values, but also display order for others. Furthermore, there is the edge of chaos, i.e. the control's precise value when the system's state switches between chaos and order.

- *Complexity involves an interaction between competition and cooperation*

  Within the cloud, resources work together to complete the job. However, they also compete for the job's sub-tasks according to their states.

From a global point of view, cloud resource management systems are concerned with many resources which collaborate directly or indirectly in order to fulfill application requirements. These resources and their interrelationships are significant in terms of the complexity occurring in such a system. From a local viewpoint, a resource in its own right may exhibit different degrees of complexity as well, which originate from internal sources (CPU, memory, disk, etc.) and/or external sources (the Jobs running on it). Therefore, the complexity presented in this study is classified into two general types: **Global System Complexity** and **Local Resource Complexity**.

4.1.2 Characteristic Of Complexity

The complexity found in cloud resource management systems has some key characteristics. It is important to understand how these characteristics affect the occurrence of complexity, either from the local resources it manages or the global system itself. However, these characteristics can act on one another or on each other. Therefore, explanations of these characteristics do not only represent the actual characteristic itself. Instead, it also emphasises the interaction and relationship among themselves.

- **Numerousness** refers to the number of cloud resources that have to be managed by the system. A large number and a high level of the resources contribute to the system's increased complexity. Changes in the number of resources that are managed by the system under any consideration directly relate to any changes in the level of complexity. In the cloud resource management problem, simply counting the number of CPU cores was sufficient for an adequate estimation of job's completion time for running jobs on a single resource. However, as the number of resources increase, it is not enough to just calculate the number of CPU cores for making resource management decision. We need to address other complexity characteristics as well in an adequate way, as we discussed in the previous chapter 3.

- **Diversity** is related to the cloud's homogeneity or heterogeneity. The resource's high/low diversity level can lead to heterogeneous/homogeneity and produces a high/low degree of complexity. Current cloud infrastructures are not yet very versatile, but heterogeneity is among the most important features which must be taken into account in making cloud management decisions. However, due to the number of factors that need to be considered (eg, CPU, memory, I/O, networking, etc.) and the use of virtualization technology, it is difficult to accurately measure heterogeneity. In this case, most current resource management solutions explicitly assume that the cloud is homogeneous, which will easily lead to poor job completion time and overall unstable cloud performance.

- **Variability** refers to the changeability state, where an event leads to possible various outcomes in the local resource or global system. In terms of the global system, the resource state changes over time (e.g. performance, availability) and leads to a change in the capacity of the system. Seen from a local resource point of view, the change in its underlying components' states (e.g. memory consumption, CPU utilisation) leads to a change in its performance. Increasing the variability leads to a higher complexity level.

- **Uncertainty** refers to all the difficulties experienced during the production of a clear picture of the resource or the system. This is caused by the lack of information. Uncertainty and complexity have a close relationship with one another. More complexity occurs when there is more uncertainty within the cloud resource management system. The uncertainty is the major difficulty in cloud computing and presents additional challenges in predicting job's completion time, which is a crucial point in many cloud resource management solutions. In most cases, the exact knowledge about the resource is not available. Therefore, it is difficult to accurately estimate the completion time of jobs, improve prediction by historical data, perform prediction correction, undertake prediction fall-back, etc.

- **Interdependency** refers to the intended or unintended relationship among cloud resource. This may lead to complexity within the management system. For instance, data required for a specific job can be partitioned or replicated onto multiple resources. These interdependent resources will not be able to perform the job without each other or without being influenced by each other. The increase of interdependence directly increases and affects complexity.

- **Variety** is related to the state of being various. In making management decisions, the states of the system (e.g. under-provision/over-provision, number of resources, order/edge of chaos/chaos, under-loaded/over-loaded) and the state of resource (e.g. high/low CPU utilisation, number of free cores, high/low memory

consumption ...) may have to be considered. This state variety represents the system or resource's dynamic behaviour. The more the states involved during decision making, the more the complexity that is introduced.

The complexity characteristics mentioned above can have close relationships with each other. In other words, one can influence the others or one can lead to the occurrence of the others. For instance, variability in the system may be created by a high level of variety or uncertainty can be caused by high density of diversity. However, the characteristics do not affect (more or less) the system with or without any interrelationships or interactions between them. Thus, generally, if these characteristics' level is reduced, the complexity will be reduced too.

4.1.3 On the Relationship Between Complexity And Entropy For Cloud Resource Management

In the previous section, we listed some of the characteristics of a complex system, evaluated the difficulty of measuring each characteristic in a cloud resource management problem and described how one characteristic will affect another. Finding a good metric with which one can measure the complexity of a system is not a trivial task. Today, most cloud resource management solutions are focused on measuring a particularly complexity characteristic while ignoring others. We argue that a good complexity measure should not solely depend on a measurement of particularly characteristic, but it must take into consideration the topological state of the complex system: from the most ordered to the most disordered.

Among many possible measures which can be used to define the state of a complex system, entropy has been by far the most popular choice. Entropy measurement is more robust, less dependent on a specific complexity characteristic, and better aligned with humans' intuitive understanding of complexity [Bonchev and Buck, 2005]. Some authors speculate that the typical relationship between complexity and entropy is unimodal: complexity values are small for small and large entropy values, but large for intermediate entropy values [Arnheim, 1974; Crutchfield and Young, 1989; Grassberger, 1986; Langton, 1990].

Entropy has many definitions and is generally divided into three categories: thermodynamic entropy, statistical entropy, and information theory entropy. In the field of computer science, information theory entropy is the most common. As the first attempt to introduce entropy measurement in a cloud resource management system, in this work, we do not introduce new complexity metrics or propose new information functions, on

which an entropy-based complexity measure could be defined. Rather, we follow the general entropy measurement and use the entropy criticism as a guiding principle of complexity measurement construction.

4.2 Complexity Management Based On Entropy Measurement

Being able to manage the increasing complexity within the cloud service resource management system is needed to better satisfy the cloud applications' QoS requirements. In order to efficiently and effectively manage complexity, it is recommended that one need to identify, measure, analyse and control complexity first. Every one of the steps mentioned above is vital to complexity management. Measuring is the most important stage since it allows for the other stages to be performed effectively [Modrak and Semanco, 2011].

4.2.1 Identifying

Identification is the first step in the process of beginning to manage the complexity in cloud resource management systems efficiently and effectively. The purpose of this step is to identify the origin of the complexity in a system and the characteristics that are related to it.

4.2.1.1 Local Activity Principle

The local activity principle was originally from electronic circuits. However, it could be mathematically formulated in an axiomatic manner without having to mention any circuit models. For a spatially-extended dynamical system that is made up of more than one identical cell, changes in the state of the cell are dictated by a specific reaction-diffusion equation and the kinetic equations related to them. In other words, changes in the local cell state are influenced by some/all of the system's other cell states and by the cell's local diffusion in some cases. Since the role of the diffusion term in the reaction-diffusion equations is only a dissipative and stabilising one, the complex phenomenon observed in the system can only originate from the cell kinetic equations [Chua, 1999]. It can be proven rigorously that if there are no locally active cell kinetic equations, complexity cannot be exhibited by the reaction-diffusion equation. A cell that possesses a local-active kinetic equation can display complex dynamics like chaos or limit cycles, even if the cells are not couple to each other. Therefore, it is no surprise that coupling

such cells could lead to an emerging pattern within the system. Thus, the cell that has a local-active kinetic equation is indeed the complexity's origin [Chua, 2005].

Definition of Local Activity : A cell is said to be locally active at a cell equilibrium point if, and only if, there exists a continuous input time function, such that at some time point there is a net energy flow out of the cell (whose initial energy was zero).

Definition of Local Passivity: A cell is said to be locally passive at a cell equilibrium point if, and only if, for all continuous input time functions, the cell remains at its initial state with zero energy.

The transistor is an typical example of a locally-active device. For the transistor, a low-power input signal can be turned into a high-power output signal. However, it is at the expense of an energy supply. Televisions, radios, or computers will not be able to function if they don't use locally-active devices like transistors. Moreover, any system that is made up of locally-active devices is considered locally active too.

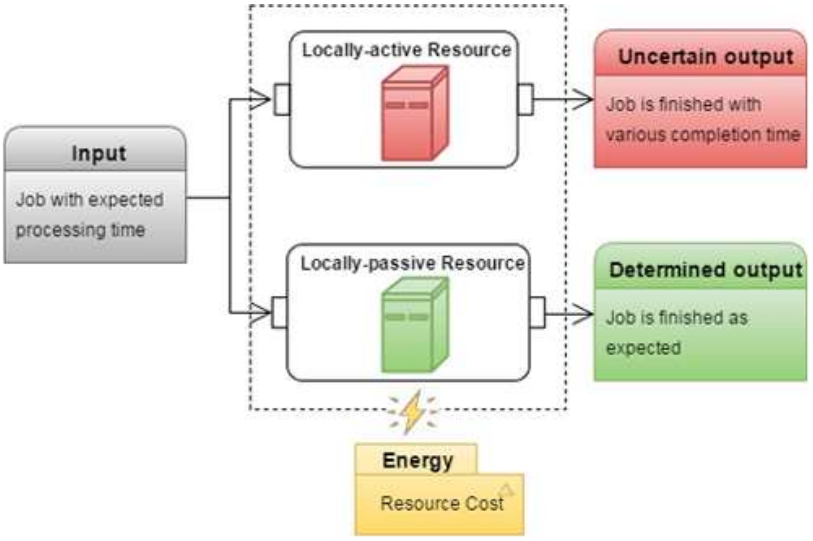

FIGURE 4.1: Locally-Active Resource Vs. Locally-Passive Resource

The **Local Activity Principle** can be easily transferred into other, non-electrical, homogeneous/heterogeneous arenas. In cloud computing, the resources are examples of locally-active devices, in which a "small" input signal (the estimated runtime of an allocated task) can convert into a "large" output signal (the actual processing time to finish the assigned task) at the expense of an energy supply (the cost of resource), as shown in Fig. 4.1. By definition, a resource is locally passive if it is not locally active, in the sense that a resource with fixed costs is guaranteed to provide an invariant performance during runtime. However, in real-world cloud systems, resources are seldom in passive in this way, but always exhibit differing degrees of local activity. For example,

on average, a physical resource is less active than a virtual resource with the same configuration and the degree of activity of a resource varies during runtime.

4.2.1.2 Origin Of Complexity: Local Active Resource

As the origin of the complexity in the system, the locally active resources have a direct impact on the complexity level of cloud resource management systems. In electronic circuits with homogeneous media, the locally active cells will put the system in a state of being at the "Edge of Chaos" [Chua, 2014] in some parameter regions, and these stand a chance of transiting to a completely chaotic state. In cloud environments, such complexity effects, caused by locally active resources, will appear more frequently. When the cloud resource management system is in a chaotic state, its performance is degraded and becomes harder to predict; it will fail to satisfactorily fulfil the QoS requirements of the application. However, in the literature, when constructing new management strategies, most of the researchers ignore the impacts of this local activity of resources on cloud resource management systems and assume the resources to be locally passive instead. So their research solutions always fail to provide satisfactory QoS when running on real world cloud environments. Some of the complexity characteristics related to the locally-active resources are as follows: **Heterogeneity**, **Dynamicity** and **Uncertainty**. More details about these characteristics can be found in the previous Chapter 2.

4.2.2 Measuring

Having identified the origin of complexity in resource management systems, the locally active resources, it is recommended, then, to provide a measurement which can determine how these resources are behaving (in relation to complexity). In fact, entropy will be used as this measurement, and, further, will be used as the measure of complexity (in relation to the definition of complexity used in this study).

4.2.2.1 Entropy Theory

Entropy is an important statistical quantity which measures the degree of disorder and the amount of wasted energy inherent to the transformation from one state to another in a system [Boltzmann, 1974]. Although the concept of entropy was originally a thermodynamic construct, it has been adapted in many other fields of study, including information theory, production planning, resource management and computer modelling and simulation [Christodoulou et al., 2009; Gan and Wirth, 2005; Hermenier et al., 2009;

Langton, 1990; Liu et al., 2008]. We will use this measure to quantify the degree of reliability which can be associated with a scheduling system under different resource allocation strategies. First, we introduce this measure in relation to a general, universal context. Given a dynamic system X with a finite mutually exclusive state variable set $S = s_1, s_2, s_3, \ldots, s_n$ with probabilities $p_1, p_2, p_3, \ldots, p_n$ respectively, entropy $H(X)$ is defined as:

$$H(X) = -\sum_{i=1}^{n} p_i * \log p_i \tag{4.1}$$

For any two mutually independent dynamic systems, A and B, with n and m states respectively, the probability of the simultaneous occurrence of the states A_i and B_j is $p_i q_j$ where p_i is the probability of state i occurring in system A and q_j is the probability of state j occurring in system B - where $1 \leq i \leq n$ and $1 \leq j \leq m$. Let the sets of states $A_i B_j$ represent another finite system designated by AB. It is easy to see that:

$$H(AB) = H(A) + H(B) \tag{4.2}$$

where $H(AB)$, $H(A)$ and $H(B)$ are the corresponding entropies of systems AB, A and B. This expression can easily be extended for an arbitrary number of mutually independent finite systems. For a system M consisting of s mutually independent sub-systems $N_1, N_2, N_3, \ldots, N_k$, the entropy is given by:

$$H(M) = -\sum_{i=1}^{k} H(N_i) \tag{4.3}$$

And the average sub-system entropy [Langton, 1990] is easily obtained by:

$$\overline{H} = \frac{H(M)}{k} \tag{4.4}$$

4.2.3 Analysis

Once a relevant measurement has been identified and made, the results of the complexity measure must be analysed. Analysing complexity values is relative to the purpose of the measurement made. A measurement can be analysed from many perspectives. For example, a complexity measure can be implemented for the purposes of :

- analysing the local activity level of resources and performing comparison among them; or
- analysing the global system to judge whether it is in a state of order or a state of chaos.

4.2.3.1 Degree Of Local Activity

The presence of local activity is the cause of breakdowns in the symmetry of homogeneous media; this insight offers a rigorous and effective tool for identifying the states of resources involved in a system. An increment in resources local activity will lead to an increment in the global system's complexity, which means the system will have a higher chance of falling into chaos.

Therefore, we introduce entropy as the quantitative measurement which can be used to compare the degree of local activity among cloud resources. The aim of measuring local activity is that of obtaining a numerical scale by which the degree of such activity of different resources can be measured. In practice, the degree of local activity is difficult to measure directly at runtime, since the cloud resources are not identical and the cloud cannot be modeled as a particular reaction-diffusion equation. However, we can judge how active a resource is through the study of its performance history in respect of CPU utilization. Generally speaking, if a resource's CPU utilization history exhibits unstable oscillations (disorder), then it has relatively high local activity and vice versa. Therefore, Entropy, as the measurement of the degree of disorder in a system, can be used to provide a quantitative measurement of the local activity degree associated with the cloud resources.

4.2.3.2 Cellular Automata

Cellular Automata (CA) can be used in modelling and simulating complex systems. The signature feature of the theory of cellular automata is the realization that "simple rules can give rise to complex behaviour". The theory was originally used to study the emergent complex behaviours of discrete dynamical networks that consist of homogeneous, local, short range interacting cells. Since the 1980s, cellular automata theory has been researched in-depth and is now widely applied in many overlapping areas, such as physical, chemical and biological systems.

A standard cellular automata scenario usually consists of four elements: cells, the state of the cells, the cells' neighbours (i.e., the relationship between the cells) and a rule for updating cells' states. In this work, we model the cloud resource management system

as a cellular automata system. In this way, the collection of cells that comprise the cellular automata consists of a number of cloud resources. The cellular automata rule that changes the resource states is defined to be the strategy we use to manage the resources (Resource allocation and Job Scheduling).

The behaviours that emerge from a Cellular Automata can be categorised into four classes:

- Spatially homogeneous state
- Sequence of simple stable or periodic structures
- Chaotic aperiodic behaviour
- Complicated localized structures, some propagating

The degree of complexity in each class can be quantitatively measured by the entropy of the global system [Langton, 1990]. In this work, we extend this Cellular Automata Entropy measurement to Cloud Resource Management Systems in order to study the complex behaviours emergent from the locally active cloud resources.

4.2.4 Controlling

Control is fundamental to management and is related to the task of taking complexity under control. Complexity not only needs to be reduced, but also, in fact, it needs to be avoided so as to prevent its existence in the future. Therefore, this step, of controlling complexity, consists of two parts: namely, reduce and avoid.

Complexity is not always easy to remove completely from a system. Thus, what needs to be considered is how to reduce complexity as much as possible. Reducing complexity is a cost-based strategy for the realisation of effective cloud resource management. Improve information sharing between cloud providers and cloud users can mitigate the existence of high complexity and help reduce costs. However, the aim of an efficient complexity management system is not only to reduce complexity levels by taking corrective actions, but also to avoid complexity by taking preventive actions for the future. Hence, the effective and efficient use of resource monitoring tools and analysis methods can help in controlling complexity in resource management.

4.3 Conclusion

Managing increasing complexity in cloud resource management systems is absolutely necessary to adequately satisfy the QoS requirements of cloud applications. In order to manage complexity effectively and efficiently, it is recommended that complexity must be defined, measured, analysed and controlled. Each of these steps is very significant to complexity management. Among these stages, measuring is the key since it facilitates the effective realisation of the other stages. In the next two chapters, I present all of these management strategies and especially concentrate on facilitating the measurability of complexity by using **Entropy Theory**.

Chapter 5

Cellular Automata Entropy: A New Cloud Resource Allocation Methodology

The content of this chapter is an extended version of the paper "**A Cost-Efficient and Reliable Resource Allocation Model Based on Cellular Automata Entropy for Cloud Project Scheduling.**" [Chen et al., 2013a] published by the *International Journal of Advanced Computer Science & Applications*.

In this chapter, the cellular automata concept is used for modelling complex multiple QoS-constrained resource management systems. Additionally, a method is presented by which the reliability of allocated cloud resources can be analysed by measuring the average resource entropy(ARE) involved. Furthermore, a **Cellular Automata Entropy-based Cloud Resource Allocation (CAE-CRA)** methodology for scheduling multiple QoS-constrained projects is proposed in order to assist in the construction, evaluation and comparison of cloud resource management strategies. Finally, the proposed methodology is implemented within the Matlab environment and verified in relation to four basic cloud resource allocation strategies, the First-Come-First-Served Algorithm (FCFS), the Round-Robin Algorithm (RR), the Min-Min Algorithm and the Max-Min Algorithm. The experimental results show that the proposed methodology can provide correct evaluations and comparisons of different resource allocation strategies and lead to the construction of more cost-efficient and reliable solutions.

5.1 Basics of Cellular Automata

We introduce a discrete lattice of cells, L, which is the state space upon which the dynamics of the CA unfolds. The discrete lattice of cells, L, is assumed to be homogeneous in that all cells bear the same properties. Further, in a one-dimensional cellular state space, the state at the discrete time t of the cell i is described by the state variable $s_i(t)$. Each cell of L is a finite automata which can assume one of a finite number of discrete values in a local value space $S \equiv \{0, 1, 2, \ldots, k-1\}$.

The generic cell, i, interacts only with a fixed number n of cells that belong to its predefined local neighbourhood N_i. At the next discrete time $t+1$, cell i updates its state, $s_i(t+1)$, according to a transition rule, $\phi : S^n \to S$, which is a function of the state variables at the time, t, of the cell n in N_i, viz:

$$s_i(t+1) = \phi[s_n(t), n \in N_i] \tag{5.1}$$

Note that the functional form of the rule is assumed to be the same everywhere in the cellular state space, i.e. there is no space index attached to ϕ. Differences between what is happening at different locations are due only to differences in the values of the state variables of the local neighbourhood, not to the update rule. The rule is also homogeneous in time. One 'iteration step' of the dynamical evolution of the CA is achieved after the simultaneous application of the rule ϕ to each cell in the lattice, L.

5.1.1 One-dimensional Cellular Automata

Consider a generic cell i of a one-dimensional lattice. The size of the neighbourhood, N_i, is defined by the radius, r, viz:

$$N_i = \{i-r, i-r+1, \ldots, i-1, i, i+1, \ldots, i+r-1, i+r\} \tag{5.2}$$

The dynamics of the system are governed by an arbitrary transition rule, $\phi : S^{2r+1} \to S$,

$$s_i(t+1) = \phi[s_{i-r}(t), \ldots, s_i(t), \ldots, s_{i+r}(t)] \tag{5.3}$$

Since cells can take any one of k values in the local value space, $S \equiv \{0, 1, 2, \ldots, k-1\}$, to completely define f one must assign a value in S to $s_i(t+1)$ for each of the k^{2r+1}

possible $(2r+1)-tuple$ configurations which can occur in the $radius-r$ neighborhood N_i of the generic cell i.

Since in correspondence to each of the k^{2r+1} possible configurations of the $radius-r$ neighborhood, N_i, any one of the k values in S can be assigned to $s_i(t+1)$, there are k^{2r+1} possible rules.

For example, let $k=2$, so that $S \equiv \{0,1\}$, and $r=1$. To define a rule one must specify the values of the generic cell i corresponding to the eight possible triplets of the neighborhood, $N_i \equiv \{i-1, i, i+1\}$. Assume an array of cells with an initial distribution of live and dead cells. Cells in the next generation of the array are calculated based on the value of the cell and the values of its left and right nearest neighbors in the current generation. If, in the following table, a live cell is represented by 1 and a dead cell by 0, then to generate the value of the cell at a particular index in the array of cellular values, you use the following Table 5.1.

TABLE 5.1: Eight Cellular Automata Rules For The Cell

Current State $[s_{i-1}(t), s_i(t), s_{i+1}(t)]$	Next Generation State	
	$s_i(t+1)$	Description
$[0,0,0]$	0	
$[0,0,1]$	0	
$[0,1,0]$	0	Dies without enough neighbours
$[0,1,1]$	1	Needs one neighbour to survive
$[1,0,0]$	0	
$[1,0,1]$	1	Two neighbours giving birth
$[1,1,0]$	1	Needs one neighbour to survive
$[1,1,1]$	0	Starved to death

The temporal evolution of this CA is obtained by:

- Specify the finite size of the array
- Specify the boundary conditions
- Specify initial distribution of live and dead cells
- Simultaneously applying the 8 CA rules to each cell of the array, in an iterative manner

Generation	Array of Cells											
	1	2	3	4	5	6	7	8	9	10	11	12
0	1	1	1	0	1	0	1	1	0	1	0	0
1	1	0	1	1	0	1	1	1	1	0	0	0
2	0	1	1	1	1	1	0	0	1	0	0	0
3	0	1	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0

FIGURE 5.1: Examples of evolution of an one-dimensional Cellular Automata.

Fig. 5.1 shows the evolution obtained by an array of 12 cells with periodic boundary conditions (e.g. $s_{13} = s_1$) and initial condition $\overrightarrow{s}(o) = [1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0]$ under the rules shown in Table 5.1. Following the exhibiting of sequences of states very different from each other, all the cells in this CA eventually die out after the 4th generations.

5.1.2 Cellular Automata Behaviour Classes

Cellular Automata may be classified with respect to the nature of their limiting behaviours. There is extensive empirical evidence that all CA rules evolving from disordered initial states fall into one of the following four basic qualitative behavioral classes [Wolfram, 1984].

- Class 1: Fixed points (the CA evolution reaches a fixed homogeneous lattice configuration in which each attains the same state value)
- Class 2: Inhomogeneous configuration or cycles (the CA evolution leads to simple stable configurations or to the emergence of periodic and separated structures)
- Class 3: Chaotic, aperiodic patterns
- Class 4: Complex, localised, propagating structures

All CA within a given class yield a qualitatively similar behaviour, regardless of the specific underlying transition rule. The behaviours of the first three classes bear a strong resemblance to those observed in continuous dynamical systems. The homogeneous final configurations occurring for the CA in class 1, for example, are essentially the same as

fixed point's attractors. Class 2 automata usually create patterns that repeat periodically, similarly to continuous limit cycles. The aperiodic, chaotic patterns emerging from class 3 automata are analogous to the strange attractors appearing in continuous dynamical systems. The statistical properties of the limit patterns and of the starting patterns are almost identical, giving rise to a kind of self-similar fractal curve. The more complicated localised structures emerging from class 4 CA do not appear to have any obvious continuous analogue. This last class of CA is capable of performing universal computation and shows a high invariability in their time development.

5.2 Project Scheduling and Cloud Resource Allocation

In this chapter, the proposed methodology has been developed under a set of assumptions:

- A project consists of a collection of tasks that have no dependencies among each other. Each task requires an amount of computing resource that is known before the task is submitted for execution, or at the time it is submitted.
- Projects needs to be completed within deadline and within a specified cost budget
- A number of cloud resources are rented in order to run a project. These resources provide a quantity of computing capacity. In this paper, computing capacities are expressed in EC2 compute units (ECU) [Amazon, 2010], which for experimental purposes were defined as 1 EC2 compute unit = 1,000,000 million instructions (per second). Hourly cost rates for one ECU were expressed in USD and were based on the EC2 pricing mode [Amazon, 2010].
- Selections of one or more scheduling strategies are available when planning the project for cloud implementation.

In static heuristics, the computing demand of each task is known prior to execution and, here, measured in ECUs. Thus, the expected execution time of a task running on a resource can be calculated by dividing the task computing demand by the resources computing capacity.

The main aim of cloud scheduling strategies is to minimize a project's completion time and cost with respect to renting a number of resources within the constraint represented by the deadline. In relation to such scheduling situations, the resource allocation problem can be defined as follows:

Let task set $T = t_1, t_2, t_3, \ldots, t_n$ be the collection of tasks in a project that is submitted to be executed on the cloud. Each task requires amounts of computing demand $cd_1, cd_2, cd_3, \ldots, cd_n$, measured in ECUs.

Let resources set $R = r_1, r_2, r_3, \ldots, r_m$ be the set of resources that are rented for scheduling the tasks. Each resource has its computing capacity which is also measured by ECU, $cc_1, cc_2, cc_3, \ldots, cc_n$. Resources are defined to be of different types according to their computing capacity [Amazon, 2010], resources type set $RT = rt_1, rt_2, rt_3, \ldots, rt_k$. The resource cost price rates for different types are $cp_1, cp_2, cp_3, \ldots, cp_k$. The project's completion time, Makespan, can be calculated as follows:

$$Makespan = max(CT_{ij}) \tag{5.4}$$

$$CT_{ij} = RT_j + ET_{ij}, 1 < i < n, 1 < j < m \tag{5.5}$$

Where CT_{ij} refers to the completion time of task i executing on resource j, ET_{ij} refers to the expected execution time of task i on resource j, and RT_j refers to the ready time of a resource j after completing the previously assigned tasks. The methodology we propose has been developed to aid decision makers to solve the following problems:

- How many of what types of resources should we rent?
- How should we schedule the multiple QoS constrained project on the rented resources?

This is so that they can achieve a cost-efficient and reliable resource allocation strategy for running the project on the cloud within deadline and within cost budget.

5.3 The Application of CA Entropy for Reliability Evaluation on Cloud Scheduling Systems

A cellular automata model can produce complex phenomenon via simple cells with simple rules; such a model has the ability to model and simulate complex systems [Von Neumann et al., 1966]. Since the 1980s, as the evolution of computer technology has progressed, cellular automata theory has attracted in-depth research and is widely applied in economics, transportation, physics, chemistry, artificial life simulations and other complex systems [Langton, 1990; Toffoli and Margolus, 1987; Wolfram, 1984].

A cellular automata consists of a regular grid of cells, each of which exists in one of a finite number of states (for instance black or white; A B, or C; 1,2,3, or 4). The grid can be in any finite number of dimensions. For each cell, a set of cells called its neighbours (usually including the cell itself) is defined relative to that specific cell. An initial state (time $t = 0$) is selected for each cell. A new generation is created according to some fixed rules that determine the new state of each cell in terms of the current state of the cell and the states of the cells in its neighbourhood. In this work, we model a cloud scheduling system's behaviour as a cellular automata (CA), specifically as a one-dimensional CA network, and then calculate the CA entropy in order to measure the degree of reliability of such a system under different scheduling rules and resource allocation strategies. For these purposes, the collection of cells that comprises the CA consists of a number of cloud resources that are rented for the running of the project (each cell of the CA corresponds to a cloud resource). The CA rules in our work are described in relation to the selected scheduling algorithm, as follows:

- First-Come-First-Served (FCFS): Tasks are executed according to the sequence in which the tasks are submitted. The first task to arrive will be scheduled on the first available resource as soon as it is submitted - and then it is removed from the queue.
- Round-Robin (RR): Schedules the first task on the first resource, the second task on the second resource, and so on, cycling through all the available resources.
- Min-Min: All the tasks in a project will be ordered by their computing demands first. The task with the minimum computing demand will be scheduled first and on the first available resource on which the completion time will be minimum - and then removed from the queue.
- Max-Min: All the tasks in a project will be ordered by their computing demands first. The task with the maximum computing demand will be scheduled first and on the first available resource on which the completion time will be minimum - and then removed from the queue.

Each resource will be in one of two performance states: Low Productivity (LP) or High Productivity (HP), which are respectively shown as Black and White on the corresponding CA grid map. The state of a resource is determined by its performance ratio under specific scheduling rules. The performance ratio of a resource (RPR) is calculated as follows:

$$RPR = \frac{Completion\ time\ for\ all\ its\ assigned\ tasks}{Completion\ time\ of\ the\ project(Makespan)} \tag{5.6}$$

If the RPR of a resource is over 50%, then it is considered to be in a High Productivity state, otherwise it is considered to be in a Low Productivity state.

The Average Resource Entropy for a CA in this context can be calculated by:

$$ARE = \sum_{i=1}^{n} \frac{-P_{LP_i} * \log P_{LP_i} - P_{HP_i} * \log P_{HP_i}}{n} \tag{5.7}$$

where n refers to the number of resources that are rented in order to run the project, and P_{LP_i} and P_{HP_i} refer to the probabilities of resources i being in the low productivity state and the high productivity state, respectively.

Reliability/unreliability is one of the basic characteristics of complex systems, and this changes with system evolution. For cloud scheduling systems, when one resource of the system suffers a loss of computing power (such losses may be caused by internal local activities or by external factors), it will fall into a low productivity state or in the worst case break down; this is called resource collapse. Such a resource collapse will influence the productivity state of all the other resources and may cause them to collapse as well, leading the scheduling system as whole to move away from an ordered condition and into a disordered/chaos condition. Along with an increase in the number of resource collapses, hierarchical up-propagation will eventually lead to the collapse of the whole scheduling system. Thus, the scheduling system will fail to deliver the project as originally planned.

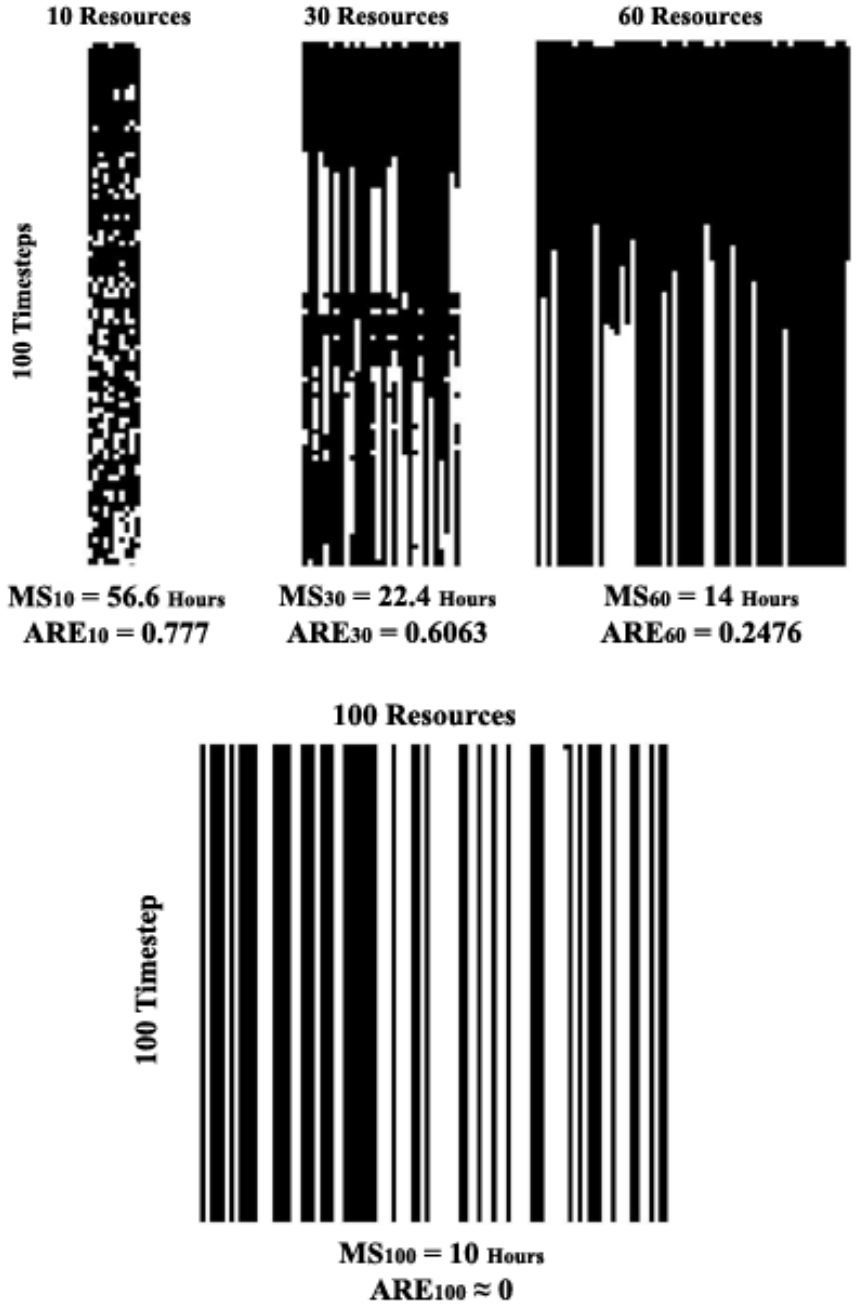

FIGURE 5.2: Scheduling Reliability: Cellular Automata Grid and Average Resource Entropy (ARE)

To evaluate the reliability of a scheduling system represented as a CA, we decrease the computing capacity of one resource by 1% at each time step, up to a total of 100 time steps; this simulates the situation where a resource degrades from full computing capacity to break down. The whole scheduling system's evolution pattern is generated and represented by CA grids. Fig. 5.2 shows some examples of grid patterns generated by a CA scheduling system using the FCFS algorithm to run a project consisting of 100 random tasks on different numbers of allocated resources. In each time step, the state of

the resource is marked in either black (High Productivity State) or white (Low Productivity State) according to the formulae 5.6. And the Average Resource Entropy(ARE) of the CA grid is calculated by formulae 5.7.

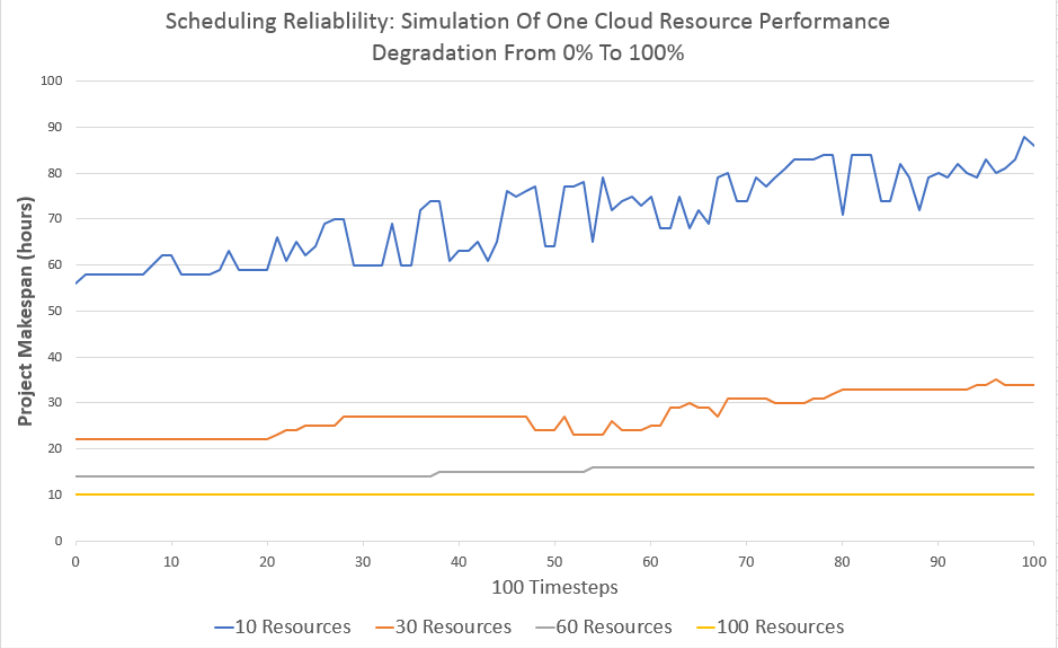

FIGURE 5.3: Scheduling Reliability Simulation: Project Makespan

From Fig. 5.3 we can see that as more resources are allocated for running jobs, the scheduling system will be more reliable, with project completion times (Makespan) more tolerant of performance degradation of individual resources, reflecting lower average resource entropy. We conclude that:

If a system is in an ordered condition, it is more reliable, and vice versa. Therefore, a systems reliability can be measured; its degree of disorder, thus the Average Resource Entropy (ARE) of a system, is a measure of its reliability.

5.4 Cellular Automata Entropy-Based Cloud Resource Allocation Methodology (CAE-CRA)

In this section, a multiple QoS-constrained Cloud Resource Allocation (CAE-CRA) methodology for scheduling projects on the cloud is proposed based on CA Entropy. The proposed methodology can be used to achieve an optimal resource allocation strategy which considers both cost-efficiency and reliability for project running in a cloud environment within deadline and cost budget constraints. The main components and the control flow of the CAE-CRA model are shown in Fig. 5.4.

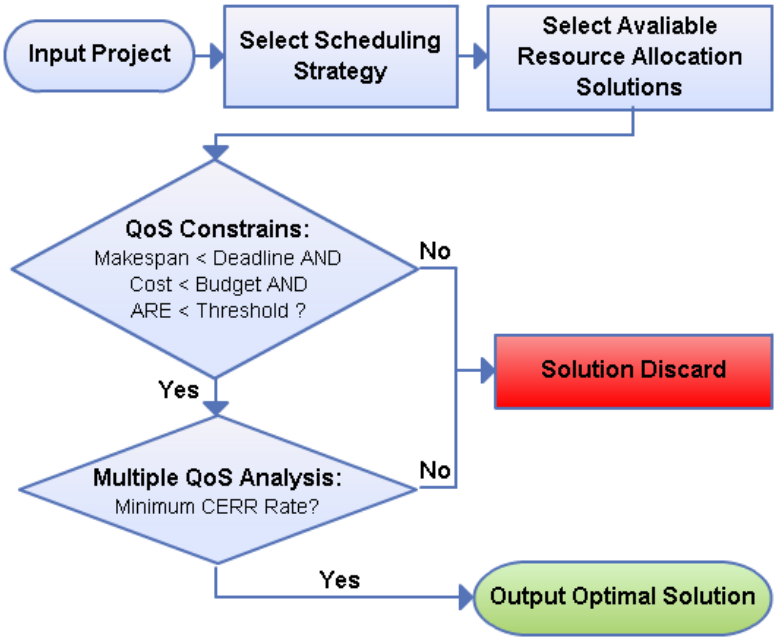

FIGURE 5.4: Flow Diagram of CAE-CRA Methodology.

The optimal resource allocation solution selected by the CAE-CRA methodology will, perforce, meet the following criteria:

- the project's multiple QoS constraints will be met: deadline, cost budget and reliability threshold; and
- an optimal Cost-Efficiency and Reliability Rate (CERR) will be maintained.

Decisions under risk assessment involve dealing with uncertainty issues, especially in areas such as cloud resource allocation issues. The classical risk assessment model in the project is $R = P * C$, which considers only the possible consequences of event which consists of the probability P and the consequence C of the event. However, the risk issue in cloud resource allocation is closely related to the uncertainty of the risk event itself. When faced with high levels of uncertainty and insufficient information, we may first consider the effect of collecting information before making a decision. Recently, Dong et al. [2016] proposed a new decision-making model for risk assessment based on entropy. They argued that risk control should include not only measures to reduce the possible consequences of the risk event, but also exploration measures to reduce uncertainty. In their approach, the risk consists of three part: probability P, consequence C and entropy H. Probability and consequence represents the hazard of risk and entropy represents the uncertainty of risk. The purpose of risk assessment is to provide a basis for risk control measures, including pre-control measures to reduce potential risk hazards and exploration measures to reduce uncertainty.

In this thesis, we use the entropy-based risk assessment model proposed by Dong et al. [2016] to guide the decision-making of cloud resource allocation problem. Motivated by the simulation results (Fig. 5.2 and Fig. 5.3) in the previous section, reducing average resource entropy should be one part of risk control and the hazard is presented by the possibility of performance degradation of a single resource. The risk assessment model considering uncertainty should obey the following principles:

- When the hazard is relatively large, we should pay the price to improve the reliability of the system, thus reduce the average resource entropy
- No matter how high the average resource entropy is, we would not take measures to improve the reliability when the hazard is small
- If the hazard and resource average entropy can be reduced to the same degree at the same cost, we would prefer choosing to reduce the hazard.

Based on the above principles, the risk level of an resource allocation strategy can be calculated by

$$R = Entropy * Hazard = ARE_n * (MS_{n-1} - MS_n) \tag{5.8}$$

where n refers to the number of resources are allocated to run the project, ARE_n refers to the average resource entropy and $(MS_{n-1} - MS_n)$ refers to the hazard of increased project makespan due to the performance degradation of one single resource.

The risk assessment involving uncertainty can also be represented in economic terms. The economic terms involve two costs. One is the Expected Cost C_E that measures cost of the project completed without any resource performance degradation as shown in Equation 5.9, and the other is Tolerance Cost C_T when the resource performance is uncertain, as shown in Equation 5.10. Where MS refers to the project's completion time and cp refers to the cost price of a resource.

$$C_E = ProjectMakespan * TotalResourceCost = MS_n * \sum_{i=1}^{n} cp_i \tag{5.9}$$

$$C_T = RiskLevel * TotalResourceCost = R * \sum_{i=1}^{n} cp_i \tag{5.10}$$

And finally, the Cost-Efficiency and Reliability Rate (CERR) can be calculated by:

$$CERR = \frac{C_E + C_T}{Cost\ Budget} \tag{5.11}$$

which reflects the risk of meeting project deadlines in the face of declining resource performance.

5.5 Experiment and Result

We implemented the proposed CAE-CRA methodology within the Matlab environment and simulated, separately, using the four basic cloud scheduling algorithms, First-Come-First-Served Algorithm (FCFS), Round-Robin Algorithm (RR), Min-Min Algorithm and Max-Min Algorithm.

5.5.1 User Case 1 - Simple Project Consisting of 10 Random Tasks

This simulated project consisted of 10 tasks with random computing demands as listed in Table 5.2. A maximum of 10 cloud resource units were available to be rented for running this project. The type of the cloud resource units available was M1 Small Instance - based on the Amazon EC2 instance types [Amazon, 2010]. The specification of M1 Small Instance is shown in Table 5.3, and the project requirements are shown in Table 5.4.

TABLE 5.2: CASE 1: PROJECT TASK SPECIFICATION

Task Specification	Task ID									
	1	2	3	4	5	6	7	8	9	10
Computing Demand (Hours of 1 ECU)	12	3	7	9	15	24	10	1	2	4

TABLE 5.3: CASE 1: CLOUD RESOURCE TYPE SPECIFICATION

Resource Type	Resource Specification		
	Computing Capacity	Price	Available
M1 Small Instance	1 ECU	\$0.115/Hour	10 Units

The experimental results, in terms of the evaluation of the four selected scheduling strategies (FCFS, RR, Min-Min, Max-Min) for all the possible resource allocations, are shown in Fig. 5.5, Fig. 5.6 and Fig. 5.7.

TABLE 5.4: CASE 1: PROJECT REQUIREMENTS

Project QoS Constrains		
Deadline	Cost Budget	Reliability
$Makespan < 35\ Hours$	$Resource\ Cost < \$20$	$ARE < 0.4$

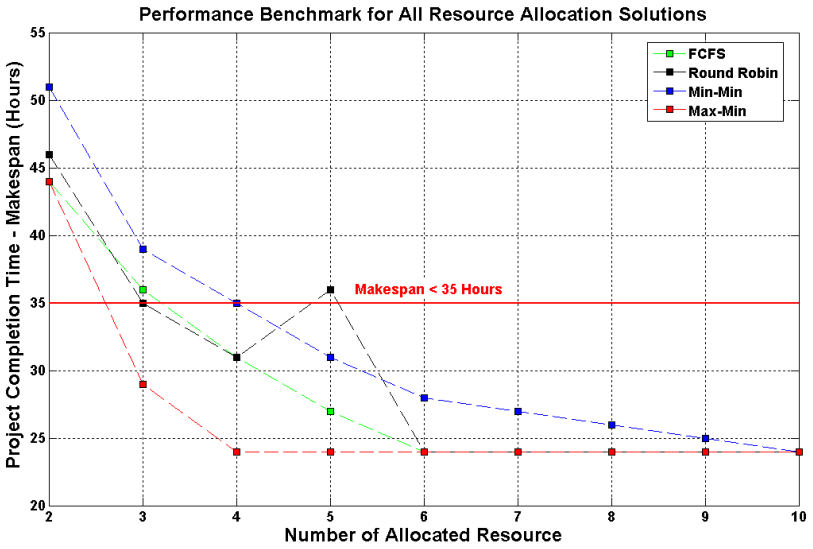

FIGURE 5.5: Performance Benchmark for All Resources Allocation Solutions (10 Tasks).

- Performance Benchmark:

  In general, the Makespan of the four scheduling strategies for the project decreased as more resources were rented. However, when the number of resources exceeded a certain limit, an investment in more resources did not improve the systems performance. It must be noted that this limit varied according to which scheduling strategy was being used. This limit was 4 resources in the case of Max-Min and 6 resources in the cases of FCFS and RR. Renting more than 5 resources resulted in limited improvement for all the scheduling strategies. However, in the case of the RR strategy, renting 5 resources decreased performance dramatically. Generally speaking, the Max-Min strategy performed better than FCFS, RR and Min-Min in most solution scenarios. The solution scenarios where less than 4 resources were rented have been discarded because of their failure to meet the deadline. The one exception to this was the solution scenario whereby 3 resources were allocated and the Max-Min scheduling strategy was used.

- Cost Benchmark:

  In most cases, the cost of the project linearly increased as more resources were rented. This was so except for the solutions which used the Max-Min strategy,

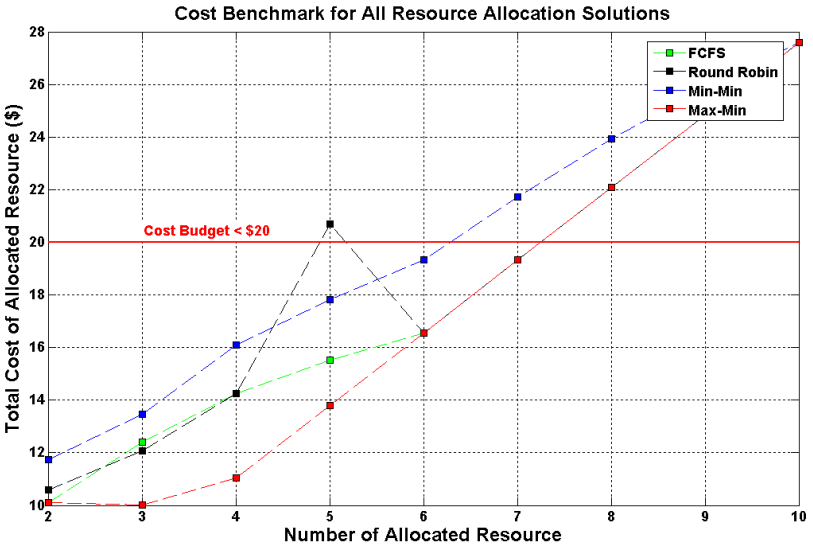

FIGURE 5.6: Cost Benchmark for All Resources Allocation Solutions (10 Tasks).

there, the costs involved when renting 2, 3 and 4 resources were similar. Under the cost budget restriction, most of the solutions which involved renting more than 6 resources were discarded.

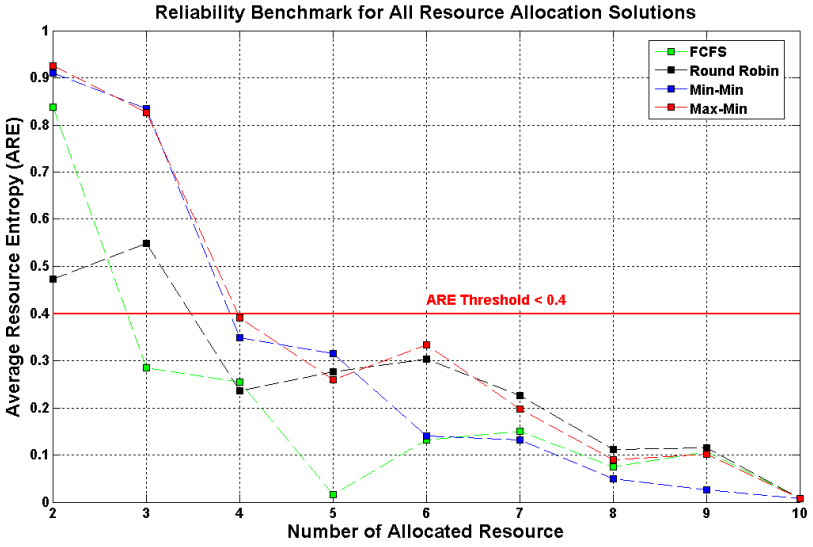

FIGURE 5.7: Reliability Benchmark for All Resources Allocation Solutions (10 Tasks).

- Reliability Benchmark:

  In general, adding a resource can improve the reliability of a system regardless of which of the three scheduling strategies has been selected. However, the reliability improvements for different scheduling strategies vary a great deal. In the case where the number of resources rented equals the number of tasks, the project gets as many resource units as it requires and the Average Resource Entropy becomes zero for all the scheduling strategies. In this case, the scheduling system has zero

entropy - indicating order and reliability. For this project, FCFS wins in relation to the reliability benchmark in most situations. Most of the solutions where less than 4 resources were rented exceeded the ARE threshold and so were discarded.

Finally, we calculate the Cost-Efficiency and Reliability Rates for all the resource allocation solutions; the CERR benchmark is shown in Fig. 5.8. We compare the CERR of all the remaining solutions which meet the project requirements as listed in Table 5.4. With the minimum CERR principle in mind, the final result and detailed performance of the optimal solution are shown in Table 5.5.

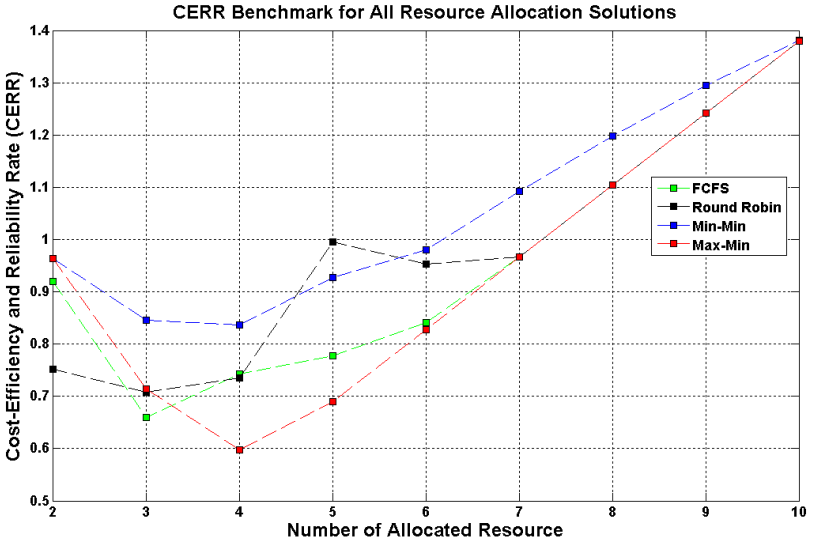

FIGURE 5.8: CERR Benchmark for All Resources Allocation Solutions (10 Tasks).

TABLE 5.5: OPTIMIZE RESOURCE ALLOCATION SOLUTIONS (10 TASKS)

Scheduling Specification	Optimal Solutions			
	$1st$	$2nd$	$3rd$	$4th$
Strategy	Max-Min	RR	FCFS	Min-Min
Rented Resources	4	4	4	4
Makespan	24 Hours	31 Hours	31 Hours	35 Hours
Cost	\$11.4	\$14.26	\$14.26	\$16.1
ARE	0.3914	0.2363	0.2550	0.3492
CERR	0.597	0.735	0.742	0.837

5.5.2 User Case 2 - Complicated Project Consists of 100 Random Tasks

In order to evaluate the robustness of the proposed CAE-CRA methodology, a more complex project consisting of 100 random tasks was simulated, and the results of this are presented in Table 5.6. The type of cloud resources and the project requirements are listed in Table 5.7 and Table 5.8. As the projects become more complicated, so it becomes harder for a decision maker to seek out an optimal solution, and thus make the project manageable. In this case, the reliability of the scheduling system is an important factor that cannot be ignored and relates directly to the risk of failing to running the project as originally planned. Thus, if the decision maker chooses an inappropriate scheduling strategy or resource allocation solution for a project, this will lead to dramatic increases in project costs, or in the worse case, a failure to complete the project within the deadline. A suitable modelling technique and the ability to accurately measure the reliability of a solution are needed for planning such large and complicated projects.

TABLE 5.6: CASE 2 : PROJECT TASK SPECIFICATION

Project Task Specification	
Total Number of Tasks	100
Total Computing Demand (Hours of 1 ECU)	5164
Maximum Computing Demand (Hours of 1 ECU)	100
Minimum Computing Demand (Hours of 1 ECU)	1
Probability distribution of Randomly generated Tasks	Normal Distribution: Many middle size tasks, and fewer big and small tasks were contained in the project

TABLE 5.7: CASE 2 : CLOUD RESOURCE TYPE SPECIFICATION

Resource Type	Resource Specification		
	Computing Capacity	Price	Available
M1 Small Instance	1 ECU	\$0.115/Hour	100Units

TABLE 5.8: CASE 2 : PROJECT REQUIREMENTS

Project QoS Constraints		
Deadline	Cost Budget	Reliability
$Makespan < 200 Hours$	$Resource\ Cost < \$800$	$ARE < 0.4$

As Fig. 5.9 shows, the performances of the four scheduling strategies are quite similar under different resource allocation solutions. However, the costs for different scheduling strategies vary a lot, as shown in Fig. 5.10. The Max-Min scheduling strategy wins in relation to the cost benchmark for most of the resource allocation solutions. The performance of the RR scheduling strategy is poor and unpredictable in this case, which means it is hard to manage and less reliable. The degree of unreliability of this strategy is correctly reflected in, and measured by, the ARE as shown in Fig. 5.11. There is no doubt that Max-Min is the optimal cost-efficient strategy for this project, and should be the one selected.

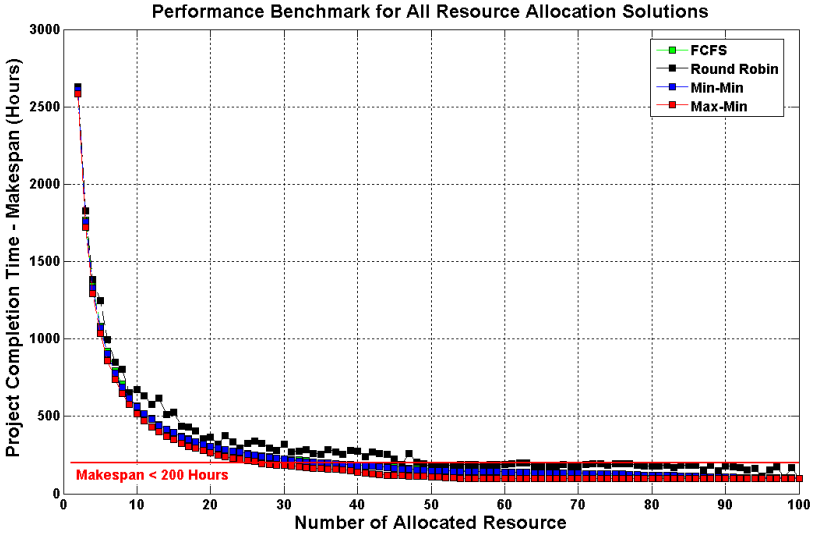

FIGURE 5.9: Performance Benchmarks for All Resources Allocation Solutions (100 Tasks).

From Fig. 5.11 we can see that the reliability of the system under the Max-Min strategy acts like a random walk as the number of allocated resources increases. At the point where 30 resources are allocated, the reliability of the system is greatly improved. After this point, the average resource entropy (ARE) of the system increases dramatically and reaches its peak at the point where 42 resources are allocated, then falls back to a more ordered state at the point of 45 resources. Overall, the ARE curve oscillates significantly and irregularly until the point where 60 resources are allocated is reached. Using most established methodologies, such fluctuating reliability in a scheduling system

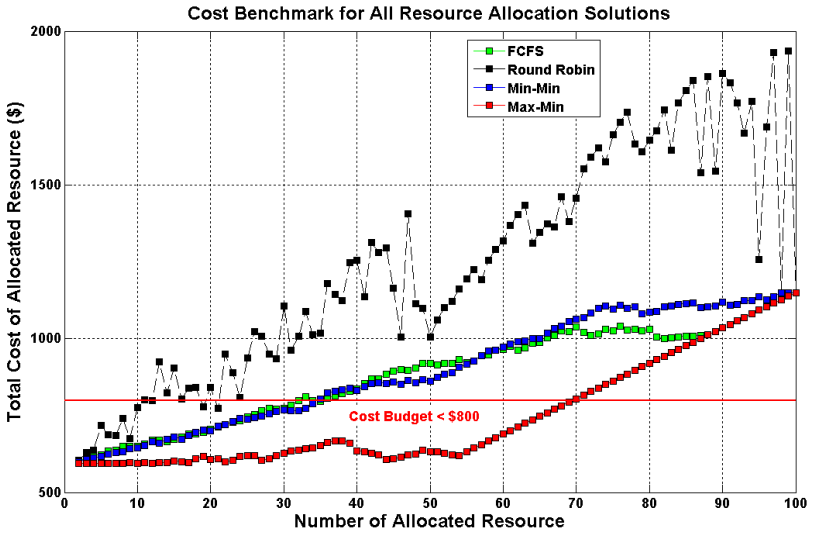

FIGURE 5.10: Cost Benchmarks for All Resources Allocation Solutions (100 Tasks).

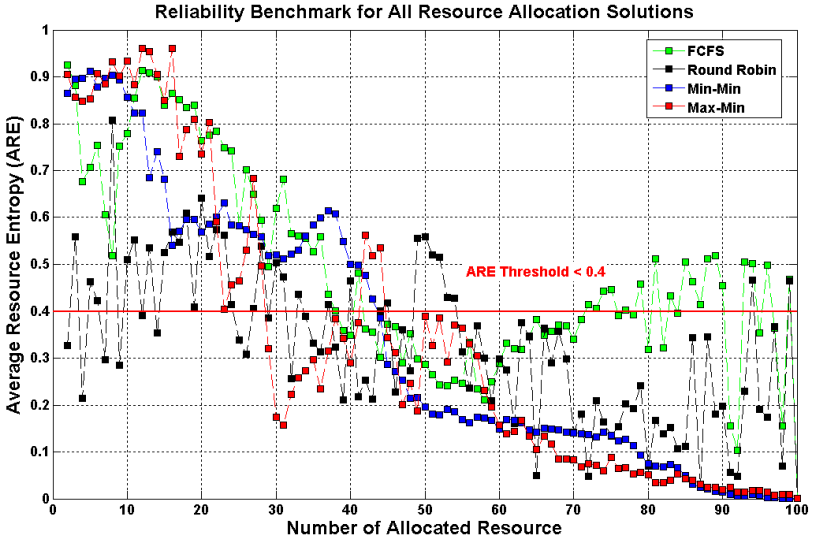

FIGURE 5.11: Reliability Benchmarks For All Resources Allocation Solutions (100 Tasks).

would difficult to model and measure, which would result in these fluctuations not being considered by the decision maker. This is especially significant when planning large and complicated projects. With our proposed CAE-CRA methodology, the above problem can be solved by the quantitative measurement of average resource entropy in the system.

Fig. 5.12 shows the CERR benchmark for all the resource allocation solutions for the project. Table 5.9 lists the comparisons of several near-optimal resource allocation solutions for running the project under the same Max-Min strategy.

From Table 5.9, some observations were drawn.

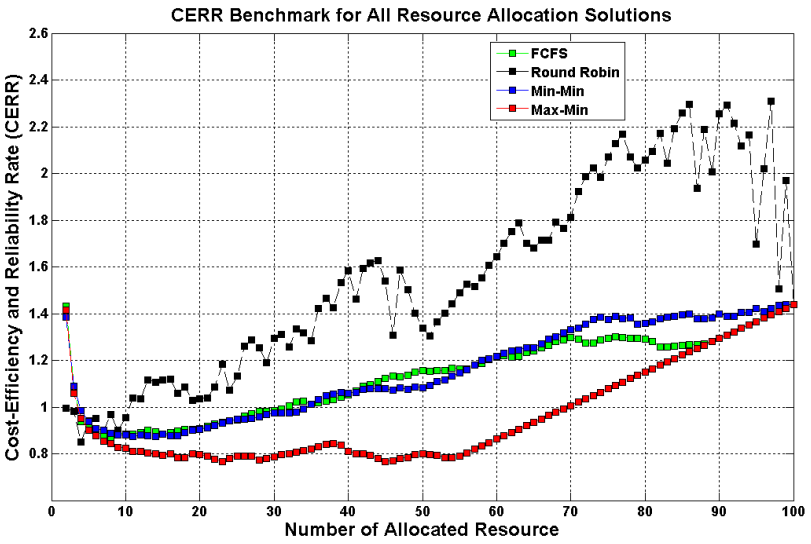

FIGURE 5.12: CERR Benchmark for All Resources Allocation Solution (100 Tasks).

TABLE 5.9: OPTIMIZE RESOURCE ALLOCATION SOLUTIONS (100 TASKS)

Max-Min Scheduling Strategy	Optimal Solutions				
	1	2	3	4	5
Rented Resources	23	30	38	44	45
Makespan (Hour)	229	182	153	120	118
Cost ($)	606	628	669	607	611
ARE	0.40	0.18	0.38	0.54	0.34
CERR	0.768	0.787	0.844	0.779	0.767

- Observation 1: Using the minimum CERR principle, solution 5 can be seen to be, and can be selected as, the optimal solution for running the project.
- Observation 2: Although solution 4 is discarded because its reliability degree (0.54) is over the ARE threshold ($ARE < 0.4$), it is still a near-optimal solution that performs almost as well as solution 5.
- Observation 3: Comparing solution 3 with solutions 2 and 4, we can see that the solution with an allocation of 38 resources for the project results in disproportionately good return on investment.
- Observation 4: The CERR value of solution 1 is close to that of solution 5, and this solution (1) has a similar cost and degree of reliability but also suffers from a huge performance difference. Since we measure the CERR strictly via the criterion of just meeting the deadline, excluding the savings in costs which could be made by savings in time (i.e., by completing before the deadline) the performance difference

is not reflected in our evaluation of the solution. In the future, this factor should be considered in our CAE-CRA methodology.

In summary, the proposed CAE-CRA methodology is capable of providing useful information and quantitative measurements for aiding the decision maker to achieve an optimal resource allocation and project scheduling solution while meeting the multiple QoS constraints.

5.6 Conclusion

Resource allocation in cloud scheduling systems is a complex problem, the solution to which requires suitable modelling and complex optimization calculations.

The experimental results show that the proposed model is able to identify both cost-efficient and reliable resource allocation solutions for projects to be run on a cloud environment, so answering the questions which someone planning to run such a project needs to ask in order to make the necessary decisions:

- How many resources do I need?
- How should I schedule the project on the resources?
- Is such a solution cost-efficient and reliable?
- Given a number of solutions, which ones are best in relation to different QoS requirements?

The CAE-CRA methodology proposed in this chapter puts forward an optimization method that is different from the established approach. It is one that is based on Cellular Automata Entropy, and, further, is based on minimizing the CERR of a scheduling system. The CERR indicates both the level of cost-efficiency and the level of reliability of the resource allocation solution - thus a low CERR will mean a more manageable project. The proposed methodology has been applied to aid the decision maker in planning multiple QoS constrained projects on a cloud environment. The experiments helped demonstrate how the CAE-CRA methodology can be implemented, how the results can be interpreted, and how a CA Entropy-based solution can be introduced into a project manager's decision-making process.

Chapter 6

Local Activity Ranking: Resource Entropy for Cloud Job Scheduling

The content of this chapter is an extended version of the paper "**Complexity Reduction: Local Activity Ranking by Resource Entropy for QoS-Aware Cloud Scheduling**" [Chen et al., 2016] published in the *2016 IEEE International Conference on Services Computing (SCC)*.

In this chapter, I first extend the "Local Activity Principle" concept with a quantitative measurement based on entropy theory. Then a new "**Entropy Scheduler**" for QoS-aware cloud scheduling is proposed, for the purpose of controlling the chaos encountered in such scheduling, based on resources Local Activity Ranking. The concept is then implemented in Apache Spark, a widely-used cloud analysis engine. Finally, experiments which demonstrate that the new "**Entropy Scheduler**" outperforms the native Spark Fair Scheduler - with server cost reduced by 23%, average response time improved by 15% - 20% and the standard deviation of the response times minimized by 30% - 45% when the Spark server is not overloaded.

6.1 Degree of Local Activity Measured By Resource Entropy

As the origin of complexity, the local activity of resources has a direct impact on the complexity level of cloud scheduling system. In electronic circuits with homogeneous media, the locally active cells will put the system in the state of being on the "Edge of Chaos" [Chua, 2014] for some parameter regions; it is possible that these will transit to a completely chaotic state. In the cloud environment, such complexity effects causied by

locally active resources appear more frequently. When the cloud scheduling system is in a chaotic state, its performance is degraded and becomes harder to predict and it fails to adequately fulfil the QoS requirements of the application. However, in the literature, most of the researchers ignore the impacts of the local activity of resources on cloud scheduling systems and assume the resources to be locally passive when constructing new schedulers. So their research solutions always fail to provide adequate QoS when running on real world cloud environments.

6.1.1 The Emergence of Complex Patterns in Cloud Scheduling: Order, Edge Of Chaos And Chaos

The principle of local activity is the cause of symmetry breaking down in homogeneous media. This offers a rigorous and effective tool to identify the states of a scheduling system (See Fig. 6.1). This tool can also be used to fine tune such states into a relatively small subset called the edge of chaos, where the emergence of complex phenomena is most likely [Chua, 2005].

The increase of local activity by resources will lead to an increase in the global scheduling system's complexity, which means the system will have a higher chance of falling into chaos. Thus, we propose the following solution to reduce the complexity and control the chaos, as shown in Fig. 6.1:

"*Avoid allocating tasks to resources with a high degree of local activity* or *allocate tasks to a set of resources with similar degrees of local activity* when making scheduling decision."

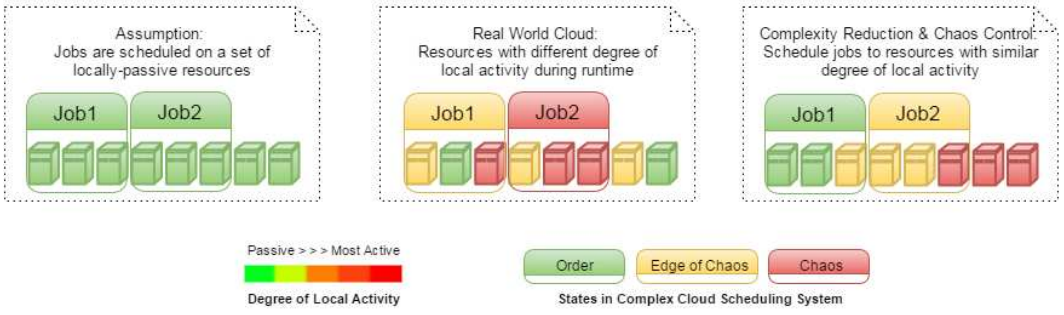

FIGURE 6.1: Complexity Reduction & Chaos Control: Resource Entropy Based Local Activity Ranking

However, this brings up another challenging problem:

"*How to provide a quantitative measurement of resource local activity during runtime in an efficient and reliable way?*"

Hence, to solve this latter problem, we introduce entropy as the quantitative measurement which can be used to compare the degree of local activity among cloud resources.

6.1.2 Entropy Measurement : Degree of Resource Local Activity

The aim of local activity measurement is to be able to obtain a numerical scale by which to compare the activity degree of different resources. In practice, the degree of local activity is difficult to obtain directly at runtime. However, we can judge how active a resource is through the study of its performance history in respect of CPU utilization. General speaking, if the resource CPU utilization history exhibits unstable oscillation (disorder), then this is the result of relatively high local activity and vice versa. Therefore, entropy, as the measurement of the degree of disorder in a system, is used to provide a quantitative measurement of the degree of local activity of different the cloud resources.

This chapter focuses on calculating the entropy value based on the resources CPU utilization history, which represents how efficiently the resource uses the CPU throughout job executions. This is highly relevant for making scheduling decisions as it is directly related to the resource's performance during runtime. The resource entropy is calculated according to the algorithms 5.

Algorithm 5 Calculate Resource Entropy

1: **Require:** $CUV \leftarrow$ CPU Utilization Vector of resource
2: **procedure** CaculateEntropy(CUV)
3: $\triangle_{cu}V \leftarrow$ Vector for changes of CPU Utilization
4: $Mean(\triangle_{cu}) \leftarrow$ Average Changes of CPU Utilization
5:
6: **if** $\triangle_{cu} \geq Mean(\triangle_{cu})$ **then**
7: $State_a \leftarrow$ Above average state
8: **else** $State_b \leftarrow$ Below average state
9:
10: $P_a \leftarrow$ Probability of $\triangle_{cu}$ in $State_a$
11: $P_b \leftarrow$ Probability of $\triangle_{cu}$ in $State_b$
12: Entropy $H(\triangle_{cu}) = -(P_a * log_2 P_a + P_b * log_2 P_b)$

The entropy measurement above has the following relationship with the degree of resource local activity:

- Entropy is a non-negative quantity: $H(\triangle_{cu}) \geq 0$, since $0 \leq P_a, P_b \leq 1$. The degree of resource local activity is proportional to the resources entropy value.
- Entropy achieves its maximum value ($H(\triangle_{cu}) = log_2(2) = 1$) when both $State_a$ and $State_b$ occur with the same probability ($P_a = P_b = 1/2$), so the resource

performance is in its most uncertain and unpredictable region, which means the degree of resource local activity is at its maximum.

- Entropy attains its minimum value $H(\triangle_{cu}) = 0$ when only one state occurs with probability 1 ($P_a = 1$ or $P_b = 1$), so the resource performance is known with complete certainty, then the degree of resource local activity is at its minimum.

6.2 Spark Entropy Scheduler : Scheduling Jobs by Resource Local Activity Ranking

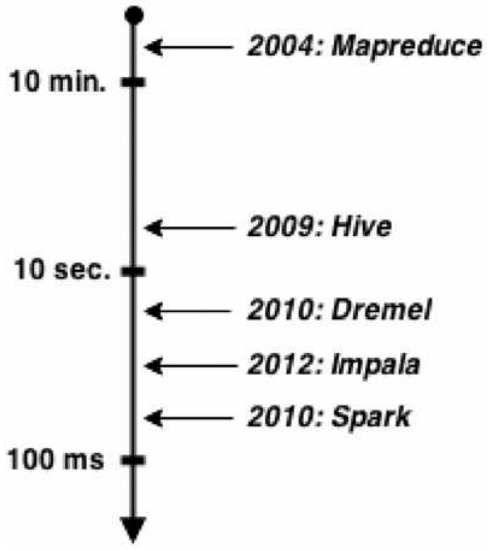

FIGURE 6.2: Cloud engines can run parallel analysis jobs with ever lower latency

Spurred by the demand for lower-latency distributed data analyses, efforts in research and industry alike have produced engines such as MapReduce [Dean and Ghemawat, 2008], Hive [Thusoo et al., 2009], Dremel [Melnik et al., 2010], Impala [Kornacker et al., 2015] and Spark [Zaharia et al., 2010] that run cloud analysis jobs across thousands of resources in a short time, as shown in Fig. 6.2. **Apache Spark** is part of the Apache Software Foundation's offering and claims speed-ups of up to 100x faster than Hadoop MapReduce in-memory, and 10 times faster on disk. The ability to bring response times into the sub-second range has enabled powerful new application developments - *Cloud Analysis as a Service* [Xu et al., 2015]. Apache Spark can provide cloud analysis query requests and responses over the HTTP web service, and supports multi-threaded querying. Fig. 6.3 illustrates the flow involved in sending an HTTP request. The Spark Web server allocates a thread to route the HTTP request to a specific cloud analysis job. Jobs are then processed with a long run global Spark Context and scheduled by the Spark Master to run on the predefined amount of Spark Workers. In such cases, user-facing

services will be able to run sophisticated parallel computation, such as language translation, voice recognition, highly personalised searches and context recommendations, on a per-query basis. However, when meeting with a high concurrency of service queries, Spark performance becomes less reliable. Spark's performance is closely tied to its job scheduler. Most of the time, we need to deploy more resources to handle an increased number of service queries, and this will cause an increase in complexity for the scheduling system.

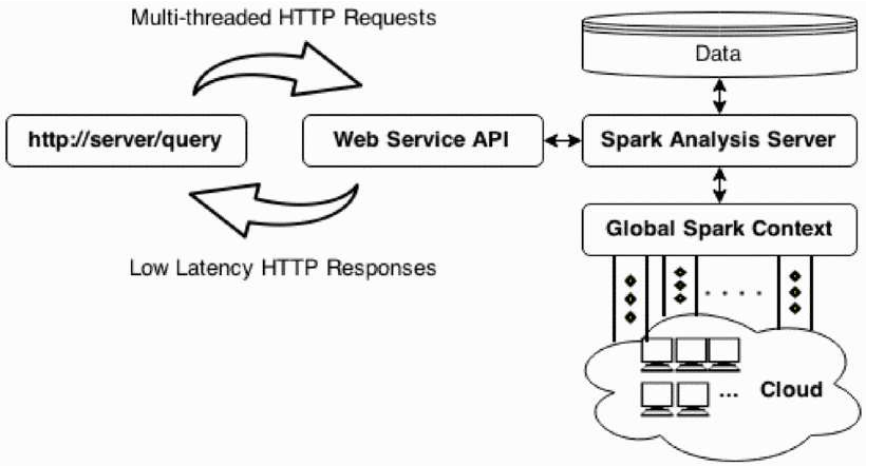

FIGURE 6.3: Apache Spark : Cloud Analysis as A Service

6.2.1 Scheduling Challenge In Spark

The Spark Context supports multi-threading and offers FIFO and FAIR scheduling options for concurrent queries. Typically, the FAIR scheduler is used for processing multiple parallel jobs simultaneously in order to minimize overall latency. The purpose of the FAIR scheduler is to assign resources to queries such that all queries get an equal share of resources over time on average. By default, the scheduler bases fairness decisions only on the number of the resources cores and its amount of memory, and assigns jobs to the resource offers via random sorting. The FAIR scheduler does not consider the core speed or current CPU utilization of the resource; these have a direct impact on the completion time of jobs. Thus, it is hard to guarantee QoS for an on-line query. If the scheduling strategy cannot provide an optimal way to guarantee QoS, it will be difficult to popularize this web service.

Scheduling low-latency parallel analysis jobs onto the heterogeneous Cloud is a challenging, multifaceted problem. Although motivated by, and designed for, the Cloud, Spark engines have not yet addressed the problem of resource scheduling for highly concurrent jobs on the heterogeneous Cloud. Spark's performance is closely tied to its job scheduler, which implicitly assumes that cloud resources are homogeneous and resources

performances do not change during run-time; it uses these assumptions to decide how to allocate jobs to resources. In practice, the homogeneity assumptions do not always hold, and the performance of resources is highly dynamic. Although the current scheduler works well in homogeneous environments, we show here that it can suffer severe performance degradation when its underlying assumptions become invalid: the performance of resources exhibits potentially uncontrollable variance and the server collapses when meeting high concurrent requests. Furthermore, we expect heterogeneous environments to become the common case as organizations often use multiple generations of hardware for building their private cloud.

6.2.2 Entropy Scheduler : A More Reliable and Efficient Solution

Optimized resource management and scheduling must take into consideration:

- The characteristics and activity of the individual resource.
- The reliability of information gain from the resource

Good job scheduling requires an awareness of resource characteristics. In the heterogeneous Cloud, the system's performance has become more sensitive to the resources which are at hand, and poor scheduling can lead to performance degradation. However, the native Spark Fair Scheduler only considers the static characteristics of resources, such as the number of available cores, while it ignores the dynamic characteristics like core performance. In such situations, jobs are unfairly scheduled on cores with differing performance, which significantly impacts on the completion time of the jobs and predictability of system performance.

In order to capture the relevant dynamic core performance characteristic, we introduce a resource activity vector (RAV) and a resource entropy level vector (REL). In the current implementation, we concentrate on the most important element of resource information, CPU utilization, which represents how efficiently the operator thread uses the CPU throughout the job's execution. This is highly relevant for making scheduling decision as it is directly related to the core's performance during run-time. To obtain the RAV values, we run a resource monitor on each worker node. The resource monitor captures the worker's CPU utilization and updates the RAV with the CPU utilization difference every second. We calculate the average change of CPU utilization (Avg) for each time period and divide the resource's history into two states (above average or below average). The REL is updated according to algorithm 6 at every heartbeat interval. Then the worker node sends the heartbeat to the master node with its current CPU utilization value and entropy level so that the latter can make informed job scheduling decisions.

Spark assumes that all the resource are homogeneous and, under Fair Schedular, randomly assigns cores to jobs. However, even in a homogeneous cloud, resources with a homogeneous setting will always running under with heterogeneous performance during run-time. Especially in the heterogeneous Cloud, such an assumption will readily result in poor job completion times and overall unstable cloud performance due to the following reasons:

- Job completion time is decided by the completion time of the slowest task in the job.
- Random core allocation will increase the chance of allocating cores with different performances to tasks inside a single job.
- Cores are not released for scheduling other jobs until the currently running job is completed. When a job is waiting for its slowest task to be completed, the computing power of the other cores, with completed tasks, is wasted.
- Monitoring and re-scheduling slow tasks (performing the speculative execution of tasks) is expensive.

In the proposed Entropy Scheduler, instead of randomly picking up resources, we first calculate the local activity ranking of all offered resources (algorithm 6), and then schedule tasks inside a job according to this ranking. Tasks are scheduled with similar ranking resources so as to improve overall QoS satisfaction and the reliability of scheduling performance.

Algorithm 6 Calculate Resource Local Activity Ranking

1: **Require:** $R_{cu} \leftarrow$ Current Resource CPU Utilization
2: **Require:** $R_e \leftarrow$ Resource Entropy
3: **Require:** $N_{cpu} \leftarrow$ Number of Available CPU cores
4: **Require:** $S_{cpu} \leftarrow$ CPU Core Clock Speed
5: **procedure** CACULATERANKING($R_{cu}, R_e, N_{cpu}, S_{cpu}$)
6: $RANK_{resource} \leftarrow$ Resource Local Activity Ranking
7: $RANK_{resource} = N_{cpu} * S_{cpu} * (1 - R_{cu}) * (1 - R_e)$

6.3 Empirical Evaluation Of Entropy Scheduler

In order to evaluate the proposed Entropy Scheduler, I conducted experiments on a private cloud with 3 heterogeneous physical resources. The resource specifications and the Spark configuration are shown on Table 6.1. A simple Spark application was deployed on the server with the ability to accept user requests to calculate π using a predefined

number of CPU cores concurrently. We used Apache Bench for the load testing of the Spark application under different schedulers (our Entropy Scheduler [Chen and Wang, 2015] and the Spark Fair Scheduler [Zaharia, 2009]). The load testing spawned a number of threads which continuously executed the same query/request. Each thread remained loaded and continued processing the query until all the threads had finished; the query response times of all the requests from every thread were used for performance comparison.

TABLE 6.1: Experimental Platform: Resource specification

Specification	Node 1	Node 2	Node 3
Spark Role	Master&Worker	Worker	Worker
CPU	Xeon 3Ghz x 2	Xeon 2.8Ghz x 2	Xeon 1.8Ghz
Cores	8	8	4
RAM	16GB	12GB	12GB

6.3.1 Experiment 1: Performance under Different Concurrent Level of HTTP Request Workload

This experiment is used to measure the average query response time and the extent to which the Entropy Scheduler and the Fair Scheduler meet QoS requirements at different concurrency levels for 100 HTTP requests workload. The results are shown in Fig. 6.5, Fig. 6.4 and Fig. 6.6.

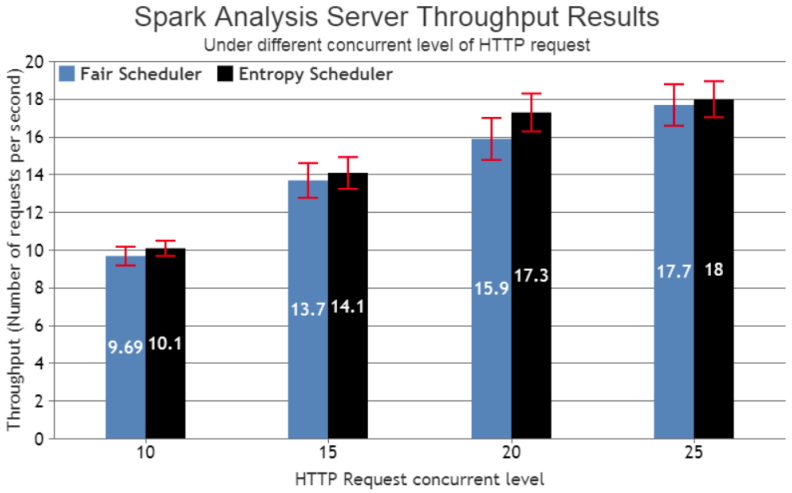

FIGURE 6.4: Experiment 1: Spark analysis server throughput result

As can be seen from Fig. 6.4, the overall performance of the two schedulers is similar. However, when the concurrency level is as low as 10 (Fig. 6.5), Entropy Scheduler performs slightly better than Fair Scheduler, where the Spark server is not overloaded.

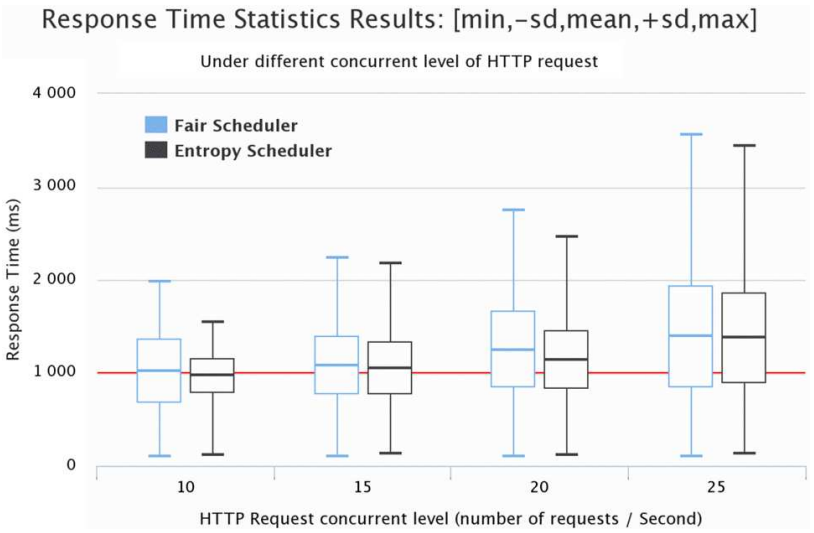

FIGURE 6.5: Experiment 1: Response time statistics result

Increasing workload concurrency poses various challenges to both schedulers. The Spark server experiences performance degradation with increasing workload concurrency. There are two main reasons behind such unstable performance:

- The loss of cloud performance and stability is due to contention and load interaction among concurrently executing queries. These effects will become worse with more complex workloads.
- The cloud, due to its parallelism and heterogeneity, is a difficult target for achieving low-latency responses since poor deployments and/or scheduling lead to performance penalties.

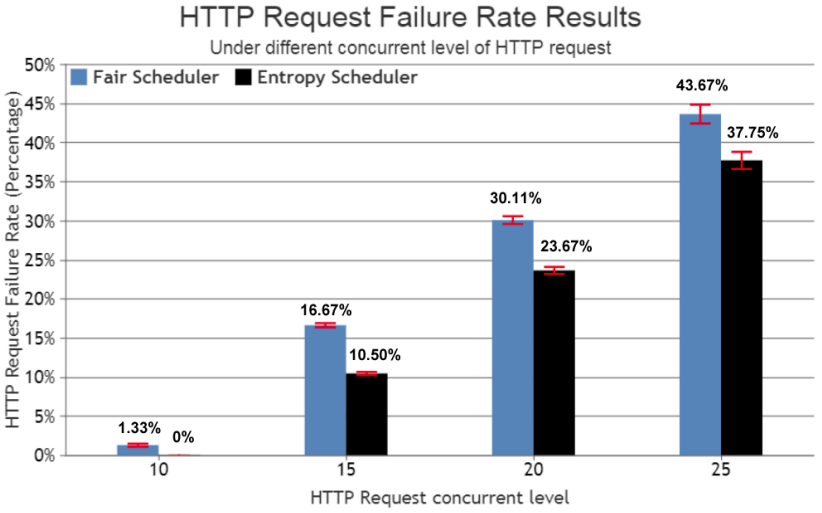

FIGURE 6.6: Experiment 1: HTTP request failure rate result

As shown in Fig. 6.6, the failure rate of the two schedulers increases as the concurrent level of requests increases. However, Entropy Scheduler significantly reduces the number of failed requests compared to the Fair Scheduler at higher concurrent level (15, 20 and 25). The t-testing (two-sample assuming unequal variances) results with 0.05 level of significance is shown in Fig. 6.7.

- $(H_0 : u_{entropy} - u_{fair} = 0)$: At the 0.05 level of significance, the sample data show there is sufficient evidence to conclude that the request failure rate that under Entropy Scheduler and Fair Scheduler is different for all concurrent level (15, 20 and 25). $H_0 : u_{entropy} - u_{fair} = 0$ is rejected.
- $(H_a : u_{entropy} - u_{fair} > 0)$: At the 0.05 level of significance, the sample data show there is sufficient evidence to conclude that the Entropy Scheduler reduces request failure rate over Fair Scheduler for all concurrent level (15, 20 and 25). $H_a : u_{entropy} - u_{fair} >= 0$ is rejected.

Concurrent Level	15		20		25	
t-Test:Failure Rate	*Entropy Scheduler*	*Fair Scheduler*	*Entropy Scheduler*	*Fair Scheduler*	*Entropy Scheduler*	*Fair Scheduler*
Mean	10.49659	16.65827	23.69948	30.07063	37.68999	43.80945
Variance	0.03753	0.07911	0.21077	0.24907	1.04398	1.33089
Observations	100	100	100	100	100	100
Hypothesized Mean Difference	0		0		0	
t Stat	-180.42022		-93.95376		-39.70939	
P(T<=t) one-tail	5.909E-202		7.154E-166		1.0901E-95	
t Critical one-tail	1.65356		1.65263		1.65271	
P(T<=t) two-tail	1.182E-201		1.431E-165		2.1802E-95	
t Critical two-tail	1.97353		1.97208		1.97220	

FIGURE 6.7: t-test result for the failure rate with Fair Scheduler and Entropy Scheduler

And Spark Server's overload point is raised under Entropy Scheduler because the request failure starts at the concurrent level of 10 with Fair Scheduler. Therefore, the experimental results show that when the Spark server is not overloaded, the Entropy Scheduler can better meet the QoS requirements than the Fair Scheduler by reducing request failure rate, thereby motivating further evaluation of larger sample workloads.

6.3.2 Experiment 2: Load Testing with 100,000 Query Requests at the Concurrent Level of 10

In this experiment, the performance of Entropy Scheduler and Fair Scheduler is evaluated under a large sample workload with a concurrency level of 10, where the Spark server is not overloaded.

TABLE 6.2: Experiment: Load testing with 100,000 query requests at the concurrent level of 10

Load Testing Result	Fair Scheduler	Entropy Scheduler
Testing Completion Time (Sec.)	951.52	732.15 (- 23%)
Throughput (Request/Sec.)	10.51	13.66 (+ 30%)
Number of failed request	75	0
Average Response Time (ms)	951	732 (- 23%)
Standard Deviation	298.9	194.7 (- 35%)

Table 6.2 compares the various aspects of load testing results produced by each scheduler. On average, in this heterogeneous cluster experiment, the Entropy Scheduler was able to shorten the load testing completion time by 23%, reduce the average response time by 23%, reduce the standard deviation of response times by 35% and improve the overall server throughput by 30% compared with the native Fair Scheduler.

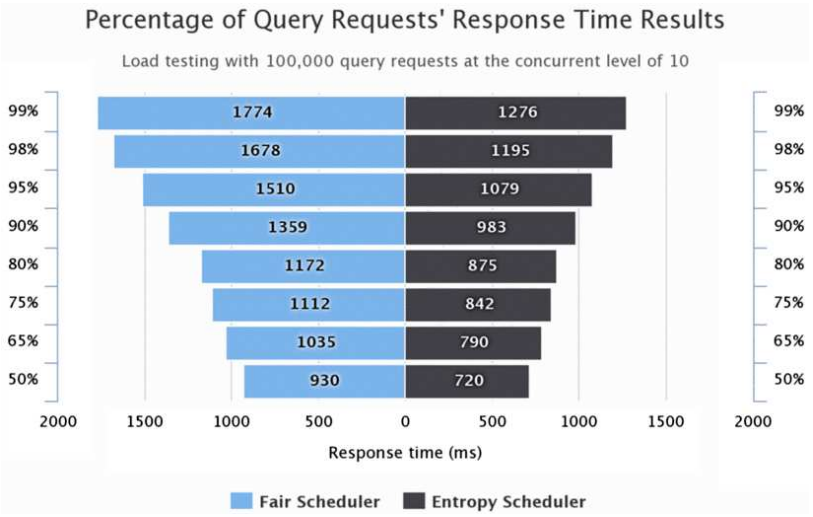

FIGURE 6.8: Experiment: Percentage of the requests served within a certain time (Million Seconds)

Fig. 6.8 indicates that 90% of queries are completed within 1 second under the Entropy Scheduler, while only 50% are completed within this time frame under the Fair Scheduler. Such results show that the Entropy Scheduler is more capable of running Cloud Analysis as a Service (CAaaS) that provide web services with a QoS guarantee.

The new Entropy Scheduler needs to be evaluated in order to compare its performance to the other exiting scheduling policies, e.g. Fair Scheduler. Based on the previous experiments, two hypotheses were investigated to determine if there was significant difference in the average response time between Entropy Scheduler and Fair Scheduler, and whether Entropy Scheduler performs better than Fair Scheduler over average response time when the Spark server is not overloaded. We set the alpha $a = 0.05$ (0.05 level of

significance) to generate the t-testing (two-Sample assuming unequal variances), which result is shown in Fig. 6.9.

	Entropy Scheduler	*Fair Scheduler*
Mean	731.75746	951.47398
Variance	37776.05696	89037.45405
Observations	100000	100000
Hypothesized Mean Difference	0	
Degree of Freedom	171908	
t Stat	-195.11019	
P(T<=t) one-tail	< 0.00001	
t Critical one-tail	1.64486	
P(T<=t) two-tail	< 0.00001	
t Critical two-tail	1.95998	

FIGURE 6.9: t-test result for the average response time with Fair Scheduler and Entropy Scheduler

- ($H_0 : u_{entropy} - u_{fair} = 0$): At the 0.05 level of significance, the sample data show there is sufficient evidence to conclude that the average response time that under Entropy Scheduler and Fair Scheduler is different. $H_0 : u_{entropy} - u_{fair} = 0$ is rejected.
- ($H_a : u_{entropy} - u_{fair} > 0$): At the 0.05 level of significance, the sample data show there is sufficient evidence to conclude that the Entropy Scheduler improves average response time over Fair Scheduler. $H_a : u_{entropy} - u_{fair} >= 0$ is rejected.

6.4 Conclusion

These experiments using only 3 resources with 20 cores are small-scale, but the experimental results provide insights for developing new schedulers based on entropy for large-scale locally active resources. The results show that the overall performance of Entropy Scheduler and Fair Scheduler is similar. However, the Entropy Scheduler performs better than the Fair Scheduler for Cloud Analysis as a Service (CAaaS) in complex cloud environments with lower concurrency level, and this may well be a starting point for future work, in which we hope to run low-latency queries with better QoS guarantees.

Complexity is an important issue that affects QoS satisfaction and brings additional challenges to the scheduling problem. In this chapter, the negative impact of complexity on deterministic cloud scheduling systems was used to motivate the development of a

new scheduler based on entropy theory to schedule tasks to resources in a real world cloud which exhibit local activity. With the results from this chapter, I provide both a concrete solution for a class of complex systems, as well as a number of ideas valuable for the analysis of the conventional engines running on the cloud.

Research on complexity has just emerged in the area of cloud scheduling. The understanding of the origin of complexity (locally-active cloud resources) and the impact of complexity (performance degradation, QoS guarantee violation and potential chaotic behaviour) offers useful insights for the discovery of the limitations of current scheduling solutions and motivates new scheduler development for complex cloud environments.

Chapter 7

Conclusion and Future Research Directions

The work presented in this PhD thesis address current research problems in Cloud Resource Management Systems and proposes an **Entropy Theory** based methodology for managing complexity in such systems. It satisfactorily fulfils the initial objectives and verifies the hypothesis presented in Chapter 1, exceeding the original expectations and producing results of scientific relevance.

7.1 Main Contributions

As shown in this thesis, it is possible to improve the performance of current Resource Management Systems by introducing **Entropy Theory** as a tool to manage the complexity present in cloud environments. The result of this work is the successful validation of the proposed solutions for resource management in cloud computing. To the best of my knowledge, this is the first work that addresses the complexity problems of cloud resource management systems. The main contributions of this thesis can be summarized as follows:

- **Contribution 1** : A simulator, **ComplexCloudSim**, was designed and implemented in order to simulate the complexity factors in cloud resource management systems, including heterogeneity, dynamicity and uncertainty. The evaluation results show that **ComplexCloudSim** is capable of validating and testing the robustness of resource management strategies in complex cloud environments. It helps us better understand the impact of complexity in real world cloud environments - which cannot be ignored when making resource management decisions.

- **Contribution 2** : **Damage Spreading Analysis** is used to study the complex patterns emerging in cloud resource management systems. The simulations prove the existence of "**Chaotic Behaviour**" in the system in some parameter regions, which may explain why most of the current resource management solutions fail to work well in real world cloud environments. Such findings motivate new ideas concerning the development of more robust cloud resource management strategies.

- **Contribution 3** : Complexity is clearly identified in cloud resource management systems; this complexity can be classified into two general types: **Global System Complexity** and **Local Resource Complexity**.

- **Contribution 4** : **Entropy** is introduced as a means to manage the complexity of cloud resource management systems, covering identification, measuring, analysing and controlling. To manage the *Global System Complexity*, a **Cellular Automata Entropy based Resource Allocation (CAE-CRA)** methodology is proposed in order to better satisfy the QoS requirements of cloud applications. To manage *Local Resource Complexity*, I firstly extend the concept of "Local Activity Principle" by introducing the **Local Activity Ranking measured by Resource Entropy** to control chaos in relation to QoS-aware cloud job scheduling. Then I implemented the **Entropy Scheduler** in the real-world cloud analysis engine, Apache Spark. Experiments showed that both of my proposed Entropy-base methods are able to improve the performance of cloud resource management systems.

7.2 Future Research Directions

Complexity is an important issue that affects QoS satisfaction, bringing additional challenges to cloud resource management system problems. In this thesis, the negative impact of complexity was used to motivate new resource management strategy developments based on **Entropy Theory**. With the results presented in this thesis, I provide both a concrete solution for a class of complex systems, as well as a number of ideas valuable for the evaluation of conventional engines running on the Cloud.

Research on complexity has just emerged in the area of cloud resource management. The understanding of the origin of complexity (locally-active cloud resource) and the impact of complexity (performance degradation, QoS guarantee violation and potential chaotic behaviour) offers useful insights into the limitations of current resource management solutions and motivates the development of new strategies in response to complex cloud environments.

The approach, in this thesis, of introducing the **Degree of Local Activity**, as measured by resource entropy, to monitor and so manage the complexity of cloud environments is the first such attempt presented in the literature related to this topic. Many problems may arise, and many issues remain open. A list of the most important ones is given in the following.

- **New Experimentation**: The proposed ideas have to be more extensively validated in order to determine the extent to which they can improve the robustness of resource management in the cloud. The validation of the ideas includes two dimensions of experimentation:

  1. The approach must be applied to more complex applications running in the Cloud in order to analyse, thoroughly, its scope and usability.
  2. The approach must be applied to more complex cloud environments by involving larger amount of resources in order to analyse its scalability.

  Such experimentation is worthy of interest because the final purpose is to integrate the framework in the daily practices of resource management for cloud applications.

- **Further Implementation**: Although the new Entropy Scheduler reduces, by a significant amount, the failure of jobs, compare to the native Spark Fair Scheduler, its jobs failure rate is still far from satisfactory. This problem may be caused by its centralized management approach. In the future, I would like to use the approach of Omega [Schwarzkopf et al., 2013], Mesos [Hindman et al., 2011], and Sparrow [Ousterhout et al., 2013] and transform the Entropy Scheduler from a centralized to a decentralized management system in order to solve such bottleneck problems when meeting with high concurrent workloads.

- **Potential Improvement**: We assume, here, that the resource management model need only take into account the CPU-related factors; however,resource management is usually influence by other factors as well, e.g. Memory, Disk I/O, Network, etc. The model can be extended to consider these factors, and this may improve it. Also the current model focuses on the resource-oriented complexity. In the future, complexity arising from other sources(workload, links between resources, the external environment) also needs to be studied.

- **Extended Analysis**: In this current work on complexity management, I focus on reducing/avoiding the complexity in order to minimize its negative effects on the cloud resource management system. However, both positive and negative effects exist, arising from increases in complexity. There exists a completely new application of the local activity principle at the so-called **Edge of Chaos** where

most complex phenomena emerge. The region termed the Edge of Chaos can mathematically rigorously be proven and confirmed to have different applications in real world systems, and so it is worth undertaking an extended analysis to draw on the advantages and avoid the disadvantage of increasing complexity.

- **Cross-disciplinary Research** : The concept of **Entropy Theory** and the **Local Activity Principle** are really fundamental in science. The concept of "**Degree of Local Activity measured by Entropy**" introduced in this thesis may inspire future applications in other domains of computer science. For example, in intrusion detection systems, the degree of local activity maybe identified as the behaviour pattern of a user and the emerging complexity pattern generated by such locally active users may be detected as intrusions. Such ideas can be easily extended to other disciplines as well, such as weather prediction, road traffic scheduling and call centre routing. I believe my work represents a step towards many fruitful research topics of the future.

Bibliography

Lucio Agostinho, Guilherme Feliciano, Leonardo Olivi, Eleri Cardozo, and Eliane Guimaraes. A Bio-inspired Approach To Provisioning Of Virtual Resources In Federated Clouds. In *IEEE 9th International Conference on Dependable, Autonomic and Secure Computing, DASC'11*, pages 598–604. IEEE, 2011.

EC Amazon. Amazon Elastic Compute Cloud (Amazon EC2), March 2010.

Bo An, Victor Lesser, David Irwin, and Michael Zink. Automated Negotiation With Decommitment For Dynamic Resource Allocation In Cloud Computing. In *9th International Conference on Autonomous Agents and Multiagent Systems*, pages 981–988. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

YARN Apache Hadoop. Yet Another Resource Negotiator. *ACM SoCC*, 2013.

Rudolf Arnheim. *Entropy And Art: An Essay On Disorder And Order*. Univ of California Press, 1974.

Enda Barrett, Enda Howley, and Jim Duggan. Applying Reinforcement Learning Towards Automating Resource Allocation And Application Scalability In The Cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

Jim Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task Scheduling Strategies For Workflow-based Applications In Grids. In *IEEE International Symposium on Cluster Computing and the Grid, CCGrid'05*, volume 2, pages 759–767. IEEE, 2005.

Stefanos Boccaletti, Celso Grebogi, Y-C Lai, H Mancini, and Diego Maza. The Control Of Chaos: Theory And Applications. *Physics Reports*, 329(3):103–197, 2000.

Ludwig Boltzmann. The Second Law Of Thermodynamics. In *Theoretical Physics and Philosophical Problems*, pages 13–32. Springer, 1974.

Danail Bonchev and Gregory A Buck. Quantitative Measures Of Network Complexity. In *Complexity in Chemistry, Biology, and Ecology*, pages 191–235. Springer, 2005.

Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, et al. A Comparison Of Eleven Static Heuristics For Mapping A Class Of Independent Tasks Onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

Marc Bux and Ulf Leser. DynamicCloudSim: Simulating Heterogeneity In Computational Clouds. In *2nd ACM Sigmod Workshop on Scalable Workflow Execution Engines and Technologies*, page 1. ACM, 2013.

Marc Bux and Ulf Leser. DynamicCloudSim: Simulating Heterogeneity In Computational Clouds. *Future Generation Computer Systems*, 46:85–99, 2015.

Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Modeling And Simulation Of Scalable Cloud Computing Environments And The CloudSim Toolkit: Challenges And Opportunities. In *International Conference on High Performance Computing & Simulation, HPCS'09*, pages 1–11. IEEE, 2009.

Eun-Kyu Byun, Yang-Suk Kee, Jin-Soo Kim, and Seungryoul Maeng. Cost Optimized Provisioning Of Elastic Resources For Application Workflows. *Future Generation Computer Systems*, 27(8):1011–1026, 2011.

Rodrigo N Calheiros, Rajiv Ranjan, César AF De Rose, and Rajkumar Buyya. CloudSim: A Novel Framework For Modeling And Simulation Of Cloud Computing Infrastructures And Services. *ArXiv Preprint ArXiv:0903.2525*, 2009.

Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. CloudSim: A Toolkit For Modeling And Simulation Of Cloud Computing Environments And Evaluation Of Resource Provisioning Algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

Huankai Chen and Frank Z Wang. Spark On Entropy: A Reliable & Efficient Scheduler For Low-latency Parallel Jobs In Heterogeneous Cloud. In *IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)*, pages 708–713. IEEE, 2015.

Huankai Chen, Frank Wang, and Na Helian. A Cost-Efficient And Reliable Resource Allocation Model Based On Cellular Automaton Entropy For Cloud Project Scheduling. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 4 (4), 2013a.

Huankai Chen, Frank Wang, Na Helian, and Gbola Akanmu. User-priority Guided Min-Min Scheduling Algorithm For Load Balancing In Cloud Computing. In *National Conference on Parallel Computing Technologies, PARCOMPTECH'13*, pages 1–8. IEEE, 2013b.

Huankai Chen, Frank Wang, Leon Chua, Matteo Migliavacca, and Na Helian. Complexity Reduction: Local Activity Ranking By Resource Entropy For QoS-aware Cloud Scheduling. In *IEEE International Conference on Services Computing (SCC)*, pages 585–592. IEEE, 2016.

Weiwei Chen and Ewa Deelman. Workflowsim: A Toolkit For Simulating Scientific Workflows In Distributed Environments. In *IEEE 8th International Conference on E-Science*, pages 1–8. IEEE, 2012.

King-Wai Chow and Bede Liu. On Mapping Signal Processing Algorithms To A Heterogeneous Multiprocessor System. In *International Conference on Acoustics, Speech, and Signal Processing, ICASSP'91*, pages 1585–1588. IEEE, 1991.

Symeon Christodoulou, Georgios Ellinas, and Pooyan Aslani. Entropy-based Scheduling Of Resource-constrained Construction Projects. *Automation in Construction*, 18(7): 919–928, 2009.

Leon Chua. *Memristor, Hodgkin-Huxley, And Edge Of Chaos*. Springer, 2014.

Leon O Chua. Passivity And Complexity. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 46(1):71–82, 1999.

Leon O Chua. Local Activity Is The Origin Of Complexity. *International Journal of Bifurcation and Chaos*, 15(11):3435–3456, 2005.

Moreno Coli and Paolo Palazzari. Real Time Pipelined System Design Through Simulated Annealing. *Journal of Systems Architecture*, 42(6):465–475, 1996.

James P Crutchfield and Karl Young. Inferring Statistical Complexity. *Physical Review Letters*, 63(2):105, 1989.

Amir Vahid Dastjerdi and Rajkumar Buyya. An Autonomous Reliability-aware Negotiation Strategy For Cloud Computing Environments. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid'12*, pages 284–291. IEEE, 2012.

Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing On Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Xin Dong, Hao Lu, Yuanpu Xia, and Ziming Xiong. Decision-making Model Under Risk Assessment Based On Entropy. *Entropy*, 18(11):404, 2016.

Hector Fernandez, Guillaume Pierre, and Thilo Kielmann. Autoscaling Web Applications In Heterogeneous Cloud Infrastructures. In *IEEE International Conference on Cloud Engineering, IC2E'14*, pages 195–204. IEEE, 2014.

Richard F Freund and Howard Jay Siegel. Guest Editor's Introduction: Heterogeneous Processing. *Computer*, 26(6):13–17, 1993.

Richard F Freund, Michael Gherrity, Stephen Ambrosius, Mark Campbell, Mike Halderman, Debra Hensgen, Elaine Keith, Taylor Kidd, Matt Kussow, John D Lima, et al. Scheduling Resources In Multi-user, Heterogeneous, Computing Environments With SmartNet. In *7th Heterogeneous Computing Workshop, HCW'98*, pages 184–199. IEEE, 1998.

H S Gan and A Wirth. Comparing Deterministic, Robust And Online Scheduling Using Entropy. *International Journal of Production Research*, 43(10):2113–2134, 2005.

Yue Gao, Yanzhi Wang, Sandeep K Gupta, and Massoud Pedram. An Energy And Deadline Aware Resource Provisioning, Scheduling And Optimization Framework For Cloud Systems. In *9th IEEE/ACM/IFIP International Conference on Hardware/-Software Codesign and System Synthesis*, page 31. IEEE Press, 2013.

Saurabh Kumar Garg and Rajkumar Buyya. NetworkCloudSim: Modelling Parallel Applications In Cloud Simulations. In *4th IEEE International Conference on Utility and Cloud Computing, UCC'11*, pages 105–113. IEEE, 2011.

Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. Workload Analysis And Demand Prediction Of Enterprise Data Center Applications. In *IEEE 10th International Symposium on Workload Characterization, IISWC'07*, pages 171–180. IEEE, 2007.

Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive Elastic Resource Scaling For Cloud Systems. In *International Conference on Network and Service Management (CNSM)*, pages 9–16. IEEE, 2010.

GOV.UK. Government Cloud Strategy: A Sub Strategy Of The Government ICT Strategy, March 2011.

Peter Grassberger. Toward A Quantitative Theory Of Self-generated Complexity. *International Journal of Theoretical Physics*, 25(9):907–938, 1986.

Rui Han, Li Guo, Moustafa M Ghanem, and Yike Guo. Lightweight Resource Scaling For Cloud Applications. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid'12*, pages 644–651. IEEE, 2012.

Rui Han, Moustafa M Ghanem, Li Guo, Yike Guo, and Michelle Osmond. Enabling Cost-aware And Adaptive Elasticity Of Multi-tier Cloud Applications. *Future Generation Computer Systems*, 32:82–98, 2014.

Tarek Hegazy. Optimization Of Resource Allocation And Leveling Using Genetic Algorithms. *Journal of construction engineering and management*, 125(3):167–175, 1999.

Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: A Consolidation Manager For Clusters. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 41–50. ACM, 2009.

Willy Herroelen and Roel Leus. Project Scheduling Under Uncertainty: Survey And Research Potentials. *European Journal of Operational Research*, 165(2):289–306, 2005.

Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A Platform For Fine-Grained Resource Sharing In The Data Center. In *8th USENIX Symposium on Networked Systems Design and Implementation, NSDI '11*, volume 11, pages 22–22, 2011.

Xia-yu Hua, Jun Zheng, and Wen-xin Hu. Ant Colony Optimization Algorithm For Computing Resource Allocation Based On Cloud Computing Environment. *Journal of East China Normal University (Natural Science)*, 1(1):127–134, 2010.

Eunji Hwang and Kyong Hoon Kim. Minimizing Cost Of Virtual Machines For Deadline-constrained Mapreduce Applications In The Cloud. In *ACM/IEEE 13th International Conference on Grid Computing, GRID'12*, pages 130–138. IEEE, 2012.

Hideo Ikeda. Chaotic Behavior In Complex Shop Scheduling. In *13th International Symposium on Advanced Intelligent Systems (ISIS) Joint 6th International Conference on Soft Computing and Intelligent Systems (SCIS)*, pages 873–876. IEEE, 2012.

Alexandru Iosup, Nezih Yigitbasi, and Dick Epema. On The Performance Variability Of Production Cloud Services. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid'11*, pages 104–113. IEEE, 2011.

Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: Fair Scheduling For Distributed Computing Clusters. In *22nd ACM symposium on Operating systems principles, SIGOPS'09*, pages 261–276. ACM, 2009.

Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical Prediction Models For Adaptive Resource Provisioning In The Cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012.

ISO. *ISO 5725-1: 1994: Accuracy (Trueness and Precision) Of Measurement Methods And Results-Part 1: General Principles And Definitions*. International Organization for Standardization Geneva, Switzerland, 1994.

Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. Optimal Cloud Resource Auto-scaling For Web Applications. In *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid'13*, pages 58–65. IEEE, 2013.

Pradeeban Kathiravelu and Luis Veiga. An Adaptive Distributed Simulator For Cloud And Mapreduce Algorithms And Architectures. In *IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC'14*, pages 79–88. IEEE, 2014.

Stuart A Kauffman. Metabolic Stability And Epigenesis In Randomly Constructed Genetic Nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.

Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. Brownout: Building More Robust Cloud Applications. In *36th International Conference on Software Engineering*, pages 700–711. ACM, 2014.

Marcel Kornacker, Alexander Behm, Victor Bittorf, Taras Bobrovytsky, Casey Ching, Alan Choi, Justin Erickson, Martin Grund, Daniel Hecht, Matthew Jacobs, et al. Impala: A Modern, Open-Source SQL Engine For Hadoop. In *7th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2015.

Kevin Lai, Lars Rasmusson, Eytan Adar, Li Zhang, and Bernardo A Huberman. Tycoon: An Implementation Of A Distributed, Market-based Resource Allocation System. *Multiagent and Grid Systems*, 1(3):169–182, 2005.

Chris G Langton. Computation At The Edge Of Chaos: Phase Transitions And Emergent Computation. *Physica D: Nonlinear Phenomena*, 42(1):12–37, 1990.

Diane Larsen-Freeman. Chaos/Complexity Science And Second Language Acquisition. *Applied Linguistics*, 18(2):141–165, 1997.

Dhinesh Babu LD and P Venkata Krishna. Honey Bee Behavior Inspired Load Balancing Of Tasks In Cloud Computing Environments. *Applied Soft Computing*, 13(5):2292–2303, 2013.

Young Choon Lee and Albert Y Zomaya. Rescheduling For Reliable Job Completion With The Support Of Clouds. *Future Generation Computer Systems*, 26(8):1192–1199, 2010.

Jiayin Li, Meikang Qiu, Zhong Ming, Gang Quan, Xiao Qin, and Zonghua Gu. Online Optimization For Scheduling Preemptable Tasks On IaaS Cloud Systems. *Journal of Parallel and Distributed Computing*, 72(5):666–677, 2012.

Xiangyu Lin and Chase Qishi Wu. On Scientific Workflow Scheduling In Clouds Under Budget Constraint. In *42nd International Conference on Parallel Processing, ICPP'13*, pages 90–99. IEEE, 2013.

Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. NIST Cloud Computing Reference Architecture. *NIST Special Publication*, 500: 292, 2011a.

Hui Liu, Dong Xu, and HuaiKou Miao. Ant Colony Optimization Based Service Flow Scheduling With Various QoS Requirements In Cloud Computing. In *1st ACIS International Symposium on Software and Network Engineering, SSNE'11*, pages 53–58. IEEE, 2011b.

Jiansheng Liu, Haining Tu, Hua Zhang, Fangchen Xia, and Daoyuan Yu. Research On Measurement Entropy-Based Of Equipment Management Complexity And Its Application In Production Planning. In *Intelligent Robotics and Applications*, pages 604–611. Springer, 2008.

Liang Liu, Hao Wang, Xue Liu, Xing Jin, Wen Bo He, Qing Bo Wang, and Ying Chen. GreenCloud: A New Architecture For Green Data Center. In *6th International Conference Industry Session on Autonomic Computing and Communications*, pages 29–38. ACM, 2009.

Maria M Lopez, Elisa Heymann, and Miquel A Senar. Analysis Of Dynamic Heuristics For Workflow Scheduling On Grid Systems. In *5th International Symposium on Parallel and Distributed Computing, ISPDC'06.*, pages 199–207. IEEE, 2006.

Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F Freund. Dynamic Mapping Of A Class Of Independent Tasks Onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59 (2):107–131, 1999.

Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Cost-and Deadline-constrained Provisioning For Scientific Workflow Ensembles In IaaS Clouds. In *International Conference on High Performance Computing, Networking, Storage and Analysis*, page 22. IEEE Computer Society Press, 2012.

Ming Mao and Marty Humphrey. Auto-scaling To Minimize Cost And Meet Application Deadlines In Cloud Workflows. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, page 49. ACM, 2011.

Jason Mars and Lingjia Tang. Whare-map: Heterogeneity In Homogeneous Warehouse-scale Computers. In *ACM SIGARCH Computer Architecture News*, number 3 in 41, pages 619–630. ACM, 2013.

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive Analysis Of Web-scale Datasets. *Proceedings of the VLDB Endowment*, 3(1-2):330–339, 2010.

Vlaidmir Modrak and Pavol Semanco. Complexity Assessment Of Supply Chains Structure: A Comparative Study. *Research Journal of Applied Sciences*, 6(7):410–415, 2011.

Alberto Núñez, Jose L Vázquez-Poletti, Agustin C Caminero, Gabriel G Castañé, Jesus Carretero, and Ignacio M Llorente. ICanCloud: A Flexible And Scalable Cloud Infrastructure Simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.

Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: Distributed, Low Latency Scheduling. In *24th ACM Symposium on Operating Systems Principles*, pages 69–84. ACM, 2013.

Deepak Poola, Saurabh Kumar Garg, Rajkumar Buyya, Yun Yang, and Kotagiri Ramamohanarao. Robust Scheduling Of Scientific Workflows With Deadline And Budget Constraints In Clouds. In *IEEE 28th International Conference on Advanced Information Networking and Applications, AINA'14*, pages 858–865. IEEE, 2014.

Ashish Raj, Kanwalpreet Kaur, Uddipan Dutta, V Venkat Sandeep, and Smitha Rao. Enhancement Of Hadoop Clusters With Virtualization Using The Capacity Scheduler. In *3rd International Conference on Services in Emerging Markets, ICSEM'12*, pages 50–57. IEEE, 2012.

Jia Rao, Yudi Wei, Jiayu Gong, and Cheng-Zhong Xu. QoS Guarantees And Service Differentiation For Dynamic Cloud Applications. *IEEE Transactions on Network and Service Management*, 10(1):43–55, 2013.

Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. Runtime Measurements In The Cloud: Observing, Analyzing, And Reducing Variance. *VLDB Endowment*, 3 (1-2):460–471, 2010.

Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: Flexible, Scalable Schedulers For Large Compute Clusters. In *8th ACM European Conference on Computer Systems*, pages 351–364. ACM, 2013.

Upendra Sharma, Prashant Shenoy, Sambit Sahu, and Anees Shaikh. A Cost-aware Elasticity Provisioning System For The Cloud. In *31st International Conference on Distributed Computing Systems (ICDCS)*, pages 559–570. IEEE, 2011.

Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: Elastic Resource Scaling For Multi-tenant Cloud Systems. In *2nd ACM Symposium on Cloud Computing*, page 5. ACM, 2011.

Sukhpal Singh and Inderveer Chana. Q-aware: Quality Of Service Based Cloud Resource Provisioning. *Computers & Electrical Engineering*, 47:138–160, 2015.

Mark Stillwell, David Schanzenbach, Frédéric Vivien, and Henri Casanova. Resource Allocation Algorithms For Virtualized Service Hosting Platforms. *Journal of Parallel and distributed Computing*, 70(9):962–974, 2010.

Mark Stillwell, Frederic Vivien, and Henri Casanova. Virtual Machine Resource Allocation For Service Hosting On Heterogeneous Distributed Platforms. In *IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*, pages 786–797. IEEE, 2012.

Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: A Warehousing Solution Over A Map-reduce Framework. *Proceedings of the VLDB Endowment*, 2(2): 1626–1629, 2009.

Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines*. MIT press, 1987.

Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective And Low-complexity Task Scheduling For Heterogeneous Computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

Nedeljko Vasic, Dejan Novakovic, Dejan Kostic, Svetozar Miucin, and Ricardo Bianchini. Accelerating Resource Allocation In Virtualized Environments Using Workload Classes And/Or Workload Signatures, March 2 2012. US Patent App. 13/411,491.

Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache Hadoop YARN: Yet Another Resource Negotiator. In *4th Annual Symposium on Cloud Computing*, page 5. ACM, 2013.

Christian Vecchiola, Rodrigo N Calheiros, Dileban Karunamoorthy, and Rajkumar Buyya. Deadline-driven Provisioning Of Resources For Scientific Applications In Hybrid Dlouds With Aneka. *Future Generation Computer Systems*, 28(1):58–65, 2012.

John Von Neumann, Arthur Walter Burks, et al. *Theory Of Self-reproducing Automata*. University of Illinois press Urbana, 1966.

Lee Wang, Howard Jay Siegel, Vwani P Roychowdhury, and Anthony A Maciejewski. Task Matching And Scheduling In Heterogeneous Computing Environments Using A Genetic-algorithm-based Approach. *Journal of Parallel and Distributed Computing*, 47(1):8–22, 1997.

Brian J Watson, Manish Marwah, Daniel Gmach, Yuan Chen, Martin Arlitt, and Zhikui Wang. Probabilistic Performance Modeling Of Virtualized Resource Allocation. In *7th International Conference on Autonomic Computing*, pages 99–108. ACM, 2010.

Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

Bhathiya Wickremasinghe et al. CloudAnalyst: A CloudSim-based Tool For Modelling And Analysis Of Large Scale Cloud Computing Environments. *MEDC Project Report*, 22(6):433–659, 2009.

Marek Wieczorek, Radu Prodan, and Thomas Fahringer. Scheduling Of Scientific Workflows In The ASKALON Grid Environment. *ACM SIGMOD Record*, 34(3):56–62, 2005.

Stephen Wolfram. Universality And Complexity In Cellular Automata. *Physica D: Nonlinear Phenomena*, 10(1):1–35, 1984.

Zhangjun Wu, Xiao Liu, Zhiwei Ni, Dong Yuan, and Yun Yang. A Market-oriented Hierarchical Scheduling Strategy In Cloud Workflow Systems. *The Journal of Supercomputing*, 63(1):256–293, 2013.

Zhen Xiao, Weijia Song, and Qi Chen. Dynamic Resource Allocation Using Virtual Machines For Cloud Computing Environment. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1107–1117, 2013.

Donna Xu, Dongyao Wu, Xiwei Xu, Liming Zhu, and Len Bass. Making Real Time Data Analytics Available As A Service. In *11th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA'15*, pages 73–82. IEEE, 2015.

Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: Precise Online QoS Management For Increased Utilization In Warehouse Scale Computers. *ACM SIGARCH Computer Architecture News*, 41(3):607–618, 2013.

Jia Yu and Rajkumar Buyya. A Budget Constrained Scheduling Of Workflow Applications On Utility Grids Using Genetic Algorithms. In *Workshop on Workflows in Support of Large-Scale Science, WORKS'06*, pages 1–10. IEEE, 2006.

Jia Yu, Rajkumar Buyya, and Chen Khong Tham. Cost-based Scheduling Of Scientific Workflow Applications On Utility Grids. In *1st International Conference on E-science and Grid Computing*, pages 8–pp. IEEE, 2005.

Yingchun Yuan, Xiaoping Li, Qian Wang, and Xia Zhu. Deadline Division-based Heuristic For Cost Optimization In Workflow Scheduling. *Information Sciences*, 179(15): 2562–2575, 2009.

Matei Zaharia. Job Scheduling With The Fair And Capacity Schedulers. *Hadoop Summit*, 9, 2009.

Matei Zaharia. The Hadoop Fair Scheduler, 2010.

Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing With Working Sets. *HotCloud*, 10:10–10, 2010.

Sharrukh Zaman and Daniel Grosu. Combinatorial Auction-based Allocation Of Virtual Machine Instances In Clouds. *Journal of Parallel and Distributed Computing*, 73(4): 495–508, 2013.

Qi Zhang, Quanyan Zhu, and Raouf Boutaba. Dynamic Resource Allocation For Spot Markets In Cloud Computing Environments. In *4th IEEE International Conference on Utility and Cloud Computing, UCC'11*, pages 178–185. IEEE, 2011.

Laiping Zhao, Yizhi Ren, and Kouichi Sakurai. Reliable Workflow Scheduling With Less Resource Redundancy. *Parallel Computing*, 39(10):567–585, 2013.

